Alessandra Bagnato (Ed.)

Security in Model-Driven Architecture

European Workshop on Security in Model Driven Architecture 2009 (SEC-MDA 2009), Enschede (The Netherlands), June 24, 2009
Proceedings

Enschede, the Netherlands, 2009
CTIT Workshop Proceedings Series WP09-06
ISSN 0929-0672

Preface

This volume contains the proceedings of the European Workshop on Security in Model Driven Architecture (SEC-MDA'09) held on 24 June 2009 in Enschede, The Netherlands, in conjunction with the Fifth European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2009).

The SEC-MDA 2009 workshop aims at helping the convergence of the academia and industry, from all the different areas that want to/might play an active role in domain of security solutions by focusing on how software security can be improved through the MDA approach.

The main discussion topics have been:

- How security specialists can capture their security expertise in form of reusable models, in particular threat and vulnerability models
- How the security requirements and goals can be traced all along the development process
- How security models and profiles can be merged with system models in different abstraction levels
- How security models can be shared and reused
- How developers can benefit from these reusable models for specification and
- How security testing can be improved through security models.
- Which are the requirements on tools to support the creation, transformation and use of security models?

There have been 8 paper submissions to this workshop, and 5 papers have been accepted for publication and oral presentations. All selected papers are of high quality, thanks to the professionalism of the authors, reviewers and program committee members.

We would like to take this opportunity to thank the people who have contributed to the SEC-MDA 2009 workshop. We wish to thank all authors and reviewers for their valuable contributions, and we wish them a successful continuation of their work in this area. We wish to thank our invited speaker Dr. Jan Jurjens from Open University for his talk on 'Model-based Security Engineering for Evolving Systems'. Finally, we thank the organization of the ECMDA-FA 2009 conference in which this workshop has been embedded.

May 2009

The Organizational Committee
Alessandra Bagnato, TXT e-solutions
Per Håkon Meland, SINTEF
Erkuden Rios, European Software Institute
Bernhard Rumpe, RWTH Aachen University
Nahid Shahmehri, Linköping University

Organisation

Workshop Chairs

Alessandra Bagnato	TXT e-solutions
Erkuden Rios	European Software Institute
Nahid Shahmehri	Linköping University

Programme Committee

Habtamu Abie, Norwegian Computing Center
Alessandra Bagnato, TXT e-solutions
Ruth Breu, University of Insbruck
Ana Cavalli, Telecom SudParis
Estíbaliz Delgado, European Software Institute
Marina Egea Gonzalez, ETH Zürich
Jan Jurjens, Computing Department, The Open University
Filippo Lanubile, Università degli Studi di Bari
Xabier Larrucea, European Software Institute
Amel Mammar, Telecom SudParis
Jason Xabier Mansell, European Software Insitute
Per Håkon Meland, SINTEF
Matteo Meucci, OWASP-Italy Chair, OWASP Testing Guide lead
Charles Bastos Rodriguez, Atos Research & Innovation Security Unit
Bernhard Rumpe, RWTH Aachen University
Nahid Shahmehri, Linköping University
Ståle Walderhaug, SINTEF

Supporting Organisations

Centre of Telematics and Information Technology, University of Twente
SHIELDS Project

Table of Contents

Model-based Security Engineering for Evolving Systems

Jan Jürjens
Open University (UK) and Microsoft Research (Cambridge)
J.Jurjens@open.ac.uk

Abstract. There is growing demand to evolve systems continuously to meet changing business needs, new regulations and policies, novel technologies and computing infrastructures. Unfortunately, the pace of required change affects developers' ability to establish and maintain desirable levels of quality of systems. Therefore, the aim of the Secure Change project is to develop techniques and tools that ensure "lifelong" compliance to security, privacy and dependability requirements of long-running evolving software systems.We present work towards addressing this challenge, namely an approach for modelbased security verification which supports change by providing a traceability link to the implementation. The approach uses a design model in the UML security extension UMLsec which can be formally verified against high-level security requirements such as secrecy and authenticity. An implementation of the specification can then be verified against the model through the traceability link. The approach supports software evolution in so far as the traceability mapping is updated when refactoring operations are regressively performed using our tool-supported refactoring technique. The proposed method has been applied to an implementation of the Internet security protocol SSL.

Software Vulnerabilities, Prevention and Detection Methods: A Review [1]

Willy Jimenez , Amel Mammar, Ana Cavalli

Telecom SudParis. 9, Rue Charles Fourier
91000 Evry, France
{name.lastname}@it-sudparis.eu

Abstract. Software is a common component of the devices or systems that form part of our actual life.These systems are usually complex and are developed by different programmers. Usually programmers make mistakes in the code which could generate software vulnerabilities. A software vulnerability is a flaw or defect in the software construction that can be exploited by an attacker in order to obtain some privileges in the system. It means the vulnerability offers a possible entry point to the system. Despite the knowledge about vulnerabilities nowadays there is still a growing tendency in the number of reported vulnerabilities, reason why software security has become an important field of research. The presence of vulnerabilities in the production of software makes necessary to have tools that can help programmers to avoid or detect them in the development of the code. Thus, in relation to our on-going research on vulnerability detection, this article presents an overview of software vulnerabilities and their prevention and detection methods.

Keywords: Software vulnerability, Prevention/Detection Methods, Testing.

1 Introduction

A software vulnerability can be seen as a flaw, weakness or even an error in the system that can be exploited by an attacker in order to alter the normal behavior of the system. Because the number of software systems increases everyday also the number of vulnerabilities. Additionally, if we consider that most of the systems are exposed to multiple users (internet) and environments (operating systems for example) then it is just a matter of time that someone can launch an attack (sequence of actions) whose consequences are unpredictable in damages and cost. Usually the goal of an attacker is to gain some privileges in the system to take control of it or to obtain valuable information for its own benefit. Then it is important for the developers and general public to know about vulnerabilities and their prevention and detection.

Under the context of the European project SHIELDS, whose main objective is to bridge the gap between security experts and software developers and thereby reduce the occurrence of security vulnerabilities; we present a review of different methods to detect and prevent software vulnerabilities as well as some well known software vulnerabilities. In most of the cases vulnerabilities are caused by improper validation of the data supplied by the user. This undesired condition is used by attackers to inject faults and malicious code into the system that allows them to run their own code and applications.

For better understanding vulnerabilities the creation of models that express the set conditions that could lead or originate them is very helpful; additionally when models are well understood they could also be used for prevention. But since it is impossible to guarantee the absence of vulnerabilities in a piece of code during its creation, then it is necessary to have methods to detect them. One possibility is security software inspections, or simply manual review of the code or related documents. Also some more automated methods for vulnerability detection can be applied, which are classified into two main categories: static, when the detection is performed without running the source code; and dynamic when the program is executed in order to detect vulnerabilities. Actually, in Telecom SudParis we are doing some research about the use of models in the detection of vulnerabilities.

[1] The research leading to these results has received funding from the European Community's Seventh Framework Program (FP 7/2007-203) under the grant agreement number 215995 (http://www.shields-project.eu/)

The organization of this paper is as follow. In section 2 we introduce some known vulnerabilities and we mention possible consequences of their exploit. Section 3 contains methods to prevent vulnerabilities. In section 4 we study vulnerability detection techniques, classified into static and dynamic according to the execution of the source code. Section 5 describes our work in progress and finally in section 6 conclusions and perspectives of this work are presented.

2 Software Vulnerabilities

As we have mentioned a vulnerable software system can be exploited by attackers and the system could be compromised, the attacker might take control of the system to damage it, to launch new attacks or obtain some privileged information that he can use for his own benefit. Considering this, it is important to know the different types of vulnerabilities, their prevention and detection in order to try to avoid their presence in the final software version of the system and then reduce the possibility of attacks and costly damages.

2.1 Examples of vulnerabilities

Most of the known vulnerabilities are associated to an incorrect manner of dealing with the inputs supplied by an user of the system, if these inputs are not correctly processed before using them inside the program they can generate unexpected behavior of the system. For instance, some known and frequent vulnerabilities are [1]:

- Buffer overflow: it occurs usually with fixed length buffers when some data is going to be written beyond the boundaries of the current defined capacity. This could lead to mal functioning of the system since the new data can corrupt the data of other buffers or processes. The buffer overflow can be used also to inject malicious code, and then the execution sequence of the program could be altered in order to execute the injected code and take control of the system.
- XSS or cross site scripting: usually associated to web applications, consists in the injection of code in the pages accessed by other users. If exploited an attacker can bypass access controls, perform phishing, identity theft or expose connections.
- SQL injection: it consists in the injection of code with the intension of exploiting the content of a database. Usually happens because the inputs are not handled correctly, the attacker can get sensitive information from the database.

However, some others common vulnerabilities that can be mentioned:

- Format string bugs: it happens when external data is given to an output function as format string argument. The output function, for instance, *printf* in C language, generates an output according to the specifications of the format string, some directives can write to memory locations, thus the attacker can use the *printf* to write malicious code and change the control flow to execute it.
- Integer overflows: can be of two different types, sign conversion bugs and arithmetic overflows. The first occurs when a signed integer is converted to an unsigned integer; while in the second the result of an arithmetic operation is an integer larger than the maximum integer and it is stored in an integer variable.

2.2 Vulnerability Modeling

Most of the vulnerabilities presented in the previous section could be prevented if the software is developed more carefully, avoiding the introduction of vulnerabilities that could be exploited by attackers. One solution is in the improvement of the knowledge and understanding of software developers about: known vulnerabilities, causes, threats, attacks and counter measures. Models are in fact adequate to implement such solution.

There is for instance a vulnerability model called *Vulnerability Cause Graph* (VCG) [2,3] which "is *a directed acyclic graph that contains one exit node representing the vulnerability being modeled, and any number of cause nodes, each of which represents a condition or event during software development that might contribute to the presence of the modeled vulnerability*". An example of a VCG representing a

known buffer overflow in *xpdf* (CVE-2005-3192) taken from [2], is shown in figure 1. In this graph we can observe the different causes and possible scenarios or sequence actions that could lead to the introduction of this kind of vulnerability. The VCG is helpful to understand what can cause the vulnerability. If causes are well understood then they could be avoided in the development process.

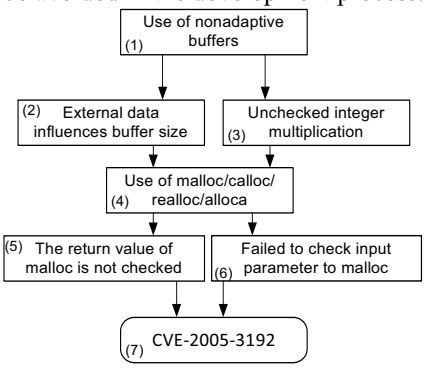


Fig. 1. Vulnerability Cause Graph

3 Preventing Software Vulnerabilities

Models are a first approach to deal with vulnerabilities and their understanding. However it is necessary to count on methods or procedures to prevent any risks related to vulnerabilities. In this section two possible vulnerability prevention methods developed in the literature are presented. The purpose is to evaluate the code during the construction process to detect any security defect and correct it on time without the need of performing intensive test at the end when the whole program is finished.

3.1 Software Inspection

The software inspection process consists in reading or visually inspecting the program code or documents in order to find any defects and correct them early in the development process. When the defect is found soon the less expensive it becomes to fix. However, a good inspection depends then on the ability and expertise of the inspector, and the kind of defects he is looking for. Usually during the software inspection, it is necessary to look for any possible defects during the security inspections. In the following sections we introduce two inspection methods that intend to codify the implicit knowledge of security experts regarding how to check for correct implementation of security goals and how to search for vulnerabilities.

3.1.1 Security Goal Indicator Trees

Security Goal Indicator Trees (SGIT) [4] focus on positive features of the software which can be verified during the inspection process. A SGIT is then a graph where the root is a security goal and its subtree are indicators or properties that can be checked for achieving that goal. However, since not all properties can be positively expressed it is possible to have also negative indicators (something that should not occur). These indicators have Boolean relations with the goal and have to be checked in order to validate the security goal. SGIT are created by security experts. A SGIT for the goal Audit Data Generation, taken from [4], is presented in figure 2, showing some dependency relations, and positive and negative indicators. Also the small box pointing to the indicator “An audit component exists” means that a specialization tree can be deployed for this indicator.

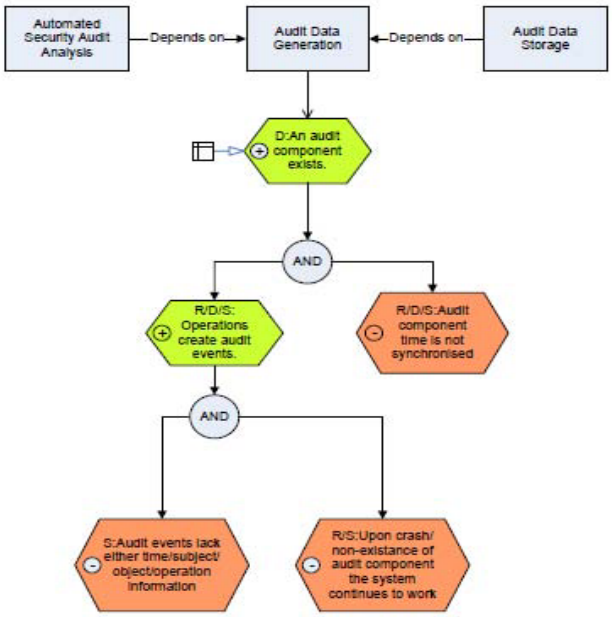


Fig. 2. Security Goal Indicator Tree

3.1.2 Vulnerability Inspection Diagram

Vulnerability Inspection Diagram (VID) is also a manual inspection introduced in [5], the purpose is to benefit developers from the knowledge and experience of security experts in the detection of problems in the development process. Thus a VID is a flowchart-like graph that guides developers to check the software to detect the presence of vulnerabilities based on the knowledge of experts. There is a specific VID for each vulnerability class.

3.2 Security Activity Graph

Security Activity Graphs (SAGs) [3,6] are also helpful in the prevention of vulnerabilities. SAGs are a graphical representation that is associated with causes in a VCG. SAGs indicate how a particular cause can be prevented following a combination of security activities during the development process. To illustrate this, in figure 3 there is a SAG [6] showing different alternatives to address the cause "Lacking design to implementation traceability".

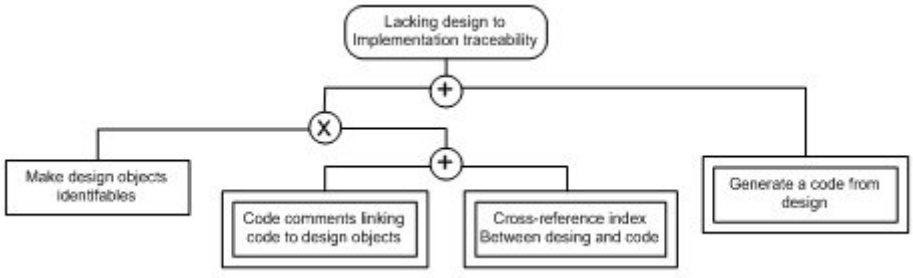


Fig. 3. Security Activity Graph

4 Detecting Software Vulnerabilities

Models and inspections are useful to understand and prevent vulnerabilities; nevertheless it is also necessary to count on tools that can be used by programmers in order to detect vulnerabilities during the process of software construction.

Some of these tools are based on static methods, thus it is not necessary to run the code to perform the detection. In the case of dynamic methods, the code is run inside a controlled environment to perform the detection or collect program traces that can be use for such purpose. In the next section we present some static and dynamic techniques to detect vulnerabilities.

4.1 Static Techniques

Static techniques are those applied directly to the source code without running the application, the objective is to evaluate or get specific information directly from the source code without executing it. There are different techniques to perform static analysis; here we mention some of them.

4.1.1 Pattern Matching

Consists in searching a "pattern" string inside the source code and give as results the number of occurrences of it. For instance if we consider C language, the pattern could be any call to possible dangerous functions (vulnerable) like "getc". Pattern matching can be implemented using a simple tool like the Unix command "grep", however this method generates much false positives because there is no analysis of the results, additionally its effectively is limited since depends on the exact writing of the strings, thus additional white spaces will limit the results.

Flawfinder also uses a more elaborated pattern matching process to find possible vulnerabilities and sort them by risk level [7]. This risk level depends on the function and on the values of the parameters of the function.

4.1.2 Lexical Analysis

Lexical analysis adds an additional step before applying a pattern match. In fact, the source code is transformed into a sequence of tokens, which are later compared with a vulnerability database in order to identify them. False positives number is still high because they do not consider the syntax or grammar of the program. The ITS4 [8] tools use lexical analysis.

4.1.3 Parsing

Parsing is more complex than lexical analysis, thus when the source code is parsed, a representation of the program is built using a parsing tree in order to analyze the syntax and the semantics of the program. For example the parsing technique is used to detect SQL command injection attacks [9].

4.1.4 Type Qualifier

Type qualifiers are used to qualify types and modify the properties of variables in the programming language, as in the case of the tool Cqual [10]. Cqual is used to specify and check properties of C programs using user-defined *type qualifiers,* which are added to the program. The modified program is analyzed to find vulnerabilities.

4.1.5 Data Flow Analysis

The purpose is to determine the possible values a variable or an expression can have during the execution of the program, specially suited for buffer overflow detection. For instance data flow analysis is used in [11]. The authors take rules describing vulnerability patterns and the source code to detect locations and paths of the pattern in the program. The process is executed in three parts: pattern matching, control and data flow and flow analyzer.

4.1.6 Taint Analysis

It is a special case of data flow analysis where any data coming from un-trusted sources, e.g. introduced by a user, is a potential problem to the system, thus it is marked as tainted. Tainted data flow is monitored because it can not reach critical functions unless it is processed and changed to untainted.

Livshits and Lam [12] propose a static analysis framework to find vulnerabilities in Java applications. They define a Tainted Object Propagation problem class to deal with improper user input validation. Java bytecode and vulnerability specifications are employed to perform a taint object propagation and find vulnerabilities using the Eclipse platform.

4.1.7 Model Checking

Model Checking is a technique to automatically test if the model of a system meets its specification and it can be used to detect vulnerabilities. Usually model checking is a complex technique because the elaboration of the model is difficult, however once obtained it is easier to test the properties of the system.

A security verification framework with multi-language support was developed [13] based on GCC compiler. Their approach uses a conventional push down system model checker for reach ability properties to verify software security properties; it is composed of three phases: security property specifications, program model extraction and property model checking, this last has as output the detected errors with execution traces.

Constraint analysis is combined with model checking [14] in order to detect buffer overflow vulnerabilities. They trace the memory size of buffer-related variables and the code instrumented with constrains assertions before the potential vulnerable points. The vulnerability can be detected with the reach ability of the assertion using model checking. They decrease the cost of model checking slicing the program.

4.2 Dynamic Techniques

In order to dynamically detect vulnerabilities it is necessary to execute the program code, and then analyze the behavior or the answers of the system and gives a verdict. In the next part we study some of the techniques to perform dynamic detection.

4.2.1 Fault Injection

Fault injection is a testing technique that introduces faults in order to test the behavior of the system, some knowledge about the system is required to generate the possible faults. With fault injection is possible to find security flaws in the system [15], in this work faults are injected into the system under test and the system behavior is observed, the failure to tolerate faults is an indicator of a potential security flaw in the system, a model is used to decide what faults to inject.

4.2.2 Fuzzing Testing

The idea of this test is to provide random data as input to the application in order to determine if the application can handle it correctly. Fuzzing testing is easier to implement than fault injection because the

test design is simpler and previous knowledge about the system to test is not always required, additionally it is limited to the entry points of the program. Web scanners are in this tool category.

Fuzzing testing can also be improved to have a better coverage of the system. For instance recording real user inputs to fill out web forms and then utilize the collected data in the fuzz testing process to better explore web applications (reach ability) [16].

4.2.3 Dynamic Taint

Similar to taint analysis, however in this case the tainted data is monitored during the execution of the program to determine its proper validation before entering sensitive functions. It enables the discovering of possible input validation problems which are reported as vulnerabilities [17].

4.2.4 Sanitization

One possibility to avoid vulnerabilities due to the use of user supply data is the implementation of new incorporated functions or custom routines whose main idea is to validate or sanitize any input from the users before using it inside a program. In [18] they present an approach using static and dynamic analysis to detect the correctness of sanitization process in web applications that could be bypass by an attacker. They use data flow techniques to identify the flows of input values from sources to sensitive sinks or the places where the value is used. Later they apply the dynamic analysis to determine the correct sanitization process.

5 Our approach

Our research in Telecom SudParis evolves around the use of models for tool-based detection of vulnerabilities. In our approach a vulnerability model, the Vulnerability Cause Graph are considered as an input in order to derive a formalism called Vulnerability Detection Condition [19], with the goal of automatically test the source code to detect vulnerabilities. The main idea behind this concept is to use the information provided by VCGs to point out in the code the use of a dangerous action under some particular conditions, for instance “it is dangerous to use unallocated memory”. The checking for vulnerabilities is performed on execution traces of the program using the TestInv tool [20]. Another tool, TestGen [20], might be adapted to generate test cases for the detection of vulnerabilities.

Another possibility considered in our research is the extension of Martins et al work [21] which uses attacks trees to generate test cases to uncover protocol vulnerabilities. The objective is to extend this work toward more general programs. Additionally we will evaluate a possible integration with the modeling tool Seamonster [22], it uses attack trees, in order to have a more powerful tool.

6 Conclusions

As we can see vulnerabilities are not a new topic on software field; however it is also noted that they still appear in the source code, thus it means that programmers still do not know how to deal with vulnerabilities. In order to help programmers to build better code we could use vulnerability cause graphs to teach them how the vulnerabilities are introduced thus they could learn to avoid the presence of vulnerabilities in their source code. However, in the mean time the source code should be inspected to guarantee there are no vulnerabilities, this method can be applied several times during the construction phase as advantage but requires specialists to perform the task as drawback.

Also a number of tools are available in order to detect vulnerabilities, some of them are based on static techniques, and it means the source code is analyzed without running the application while on dynamic techniques it is necessary to run it. The selection of the tools is related to the type of application to evaluate, the programming language and the type of vulnerability to detect. The static techniques cover all possible execution paths but require the source code while dynamic techniques have the difficulty of requiring the preparation of test cases and the possibility that not all paths in the program are covered, but the advantage

that the problems if any, are found in the running code. Dynamic techniques have also less false positives than statics.

Finally, our current research intends to create new vulnerability detection methods based on models. In this manner we could guarantee the reusability of the tests cases and facilitate the transformation of these formal representations into the specific programming language of the tool used to perform the vulnerability detection.

References

1. S. Christey. Unforgivable Vulnerabilities. The MITRE Corpotarion.2007.
2. D. Byers, S. Ardi, N. Shahmehri, C. Duma. Modeling Software Vulnerabilities with Vulnerability Cause Graphs. In Proceedings of the International Conference on Software Maintenance, Philadelphia, PA, USA, 2006.
3. S. Ardi, D. Byers, and N. Shahmehri. Towards a structured unified process for software security. In Proceedings of the ICSE 2006 Workshop on Software Engineering for Secure Software (SESS06), Shanghai, China, 2006.
4. H. Peine, M. Jawurek, S. Mandel. Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection. HASE, pp.9-18, 2008 11th IEEE High Assurance Systems Engineering Symposium.
5. SHIELDS Project Consortium. D2.1 Formalism definitions and representation schemata. SHIELDS Project Deliverable D2.1. http://www.shields-project.eu/.
6. D. Byers, N. Shahmehri. A Cause-Based Approach to Preventing Software Vulnerabilities. ARES, pp.276-283, 2008 Third International Conference on Availability, Reliability and Security.
7. D. Wheeler. Flawfinder, April 2007. http://www.dwheeler.com/flawfinder/.
8. ITS4: Software Security Tool. http://www.cigital.com/its4/.
9. Z. Su, G. Wassermann. The Essence of Command Injection Attacks in Web Applications. Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp.372-382, 2006.
10. Cqual, A tool for adding type qualifiers to C. http://www.cs.umd.edu/~jfoster/cqual/.
11. H. Kim, T. Choi, S. Jung, H. Kim, O. Lee, K. Doh. Applying Dataflow Analysis to Detecting Software Vulnerabilities. ICACT, pp.255-258. 10th International Conference on Advanced Communication Technology, 2008.
12. V. Livshits and M. Lam. Finding Security Vulnerabilities in Java Applications with Static Analysis in Proceedings of the 14th conference on USENIX Security Symposium - Volume 14, 2005, pp.18.
13. R. Hadjidj, X. Yang, S. Tlili, M. Debbabi. Model-Checking for Software Vulnerabilities Detection with Multi-Language Support. PST, pp.133-142, 2008 Sixth Annual Conference on Privacy, Security and Trust.
14. L. Wang, Q. Zhang, P. Zhao. Automated Detection of Code Vulnerabilities Based on Program Analysis and Model Checking. SCAM, pp.165-173, 2008 Eight IEEE International Working Conference on Source Code Analysis and Manipulation.
15. W. Du, A. Mathur. Vulnerability Testing of Software System Using Fault Injection. in Proceeding of the International Conference on Dependable Systems and Networks (DSN 2000), Workshop On Dependability Versus Malicious Faults.
16. S. McAllister, E. Kirda, C. Kruegel. Expanding Human Interactions for In-Depth Testing of Web Applications. RAID 2008, 11th Symposium on Recent Advances in Intrusion Detection.
17. B. Chess, J. West. Dynamic Taint Propagation: Finding Vulnerabilities without Attacking. Information Security Technical Report. Volume 13, Issue 1, 2008, Pages 33-39.
18. D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel§, and G. Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. SP, pp.387-401, 2008 IEEE Symposium on Security and Privacy.
19. A. Mammar, A. Cavalli, E. Montes de Oca, S. Ardi, D. Byers, N. Shahmehri. Modélisation et Détection Formelles de Vulnérabilités Logicielles par le Test Passif. 4ème conférence sur la Sécurité des Architectures réseaux et des Systèmes d'information. June 2009.
20. A. Cavalli, E. Montes De Oca, W. Mallouli, M. Lallali. Two Complementary Tools for the Formal Testing of Distributed Systems with Time Constraints, The 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2008).
21. E. Martins, A. Morais, A. Cavalli. Generating Attack Scenarios for the Validation of Security Protocol Implementations. The 2nd Brazilian Workshop on Systematic and Automated Software Testing (SBES 2008 - SAST), October 2008, Brazil.
22. Meland P., Spampinato D., Hagen E., Baadshaug E., Krister K., Velle K. (2008). SeaMonster: Providing tool support for security modeling. NISK 2008. National Conference on Information Security. November 2008.

A qualitative evaluation of model-based security activities for software development

Erkuden Rios[1], Per Håkon Meland[2], Shanai Ardi[3],
Alessandra Bagnato[4], Jostein Jensen[2] , Wissam Mallouli[5],
Fabio Raiteri[4], Txus Sanchez[1], Inger Anne Tøndel[2], Bachar Wehbi[5]

[1] European Software Institute, Parque Tecnológico 204, 48170 Zamudio. Spain
{erkuden.rios, jesus.sanchez}@esi.es,
[2] SINTEF ICT, Software Engineering, Safety and Security, NO-7465 Trondheim, Norway
{per.h.meland, jostein.jensen, inger.a.tondel}@sintef.no,
[3] Department of computer and information science
Linköpings universitet, SE-58183, Linköping, Sweden
{shaar}@ida.liu.se,
[4] TXT e-solutions S.p.A, Via Al Ponte Reale 5, 16100 Genoa, Italy
{alessandra.bagnato, fabio.raiteri}@txt.it,
[5] Montimage, 39 rue Bobillot, 75013 Paris, France
{wissam.mallouli, bachar.wehbi}@montimage.fr

Abstract. Most of the reoccurring types of security problems can be solved by known mitigations in most software products, preferably as early as possible during development. Representing mitigation knowledge in form of reusable security models will help developers in improving software security and learning from past mistakes. This paper explains six model-based security activities that can be integrated with most existing development processes, along with the methods and results of a qualitative evaluation involving software developers from the industry. The evaluation includes semi-structured interviews and questionnaires based on the Technology Acceptance Model (TAM).

Keywords: model-based security, software development, security engineering, qualitative evaluation.

1 Introduction

As stated by Noopur Davis [1]: "*...over 90 % of software security vulnerabilities are caused by known software defect types. [...] the top ten causes account for about 75 % of all vulnerabilities.*" Therefore, in most systems it is possible to substantially improve security by focusing on common security problems that can be solved in similar ways in most software products. Our approach is to transform up-to-date information on security problems and ways to mitigate them in the form of reusable security models. These models are intended to help developers improve software security and learn from past mistakes, and they can be accessed from within development tools.

The purpose of this paper is to present a set of model-based security activities with their supporting modeling formalisms. These activities were subjected to a qualitative evaluation which we also explain and show the results from. All the work has been performed in the context of the EU project SHIELDS [2], which is about reducing known security vulnerabilities during software development by sharing security models through a centralized repository [3].

The qualitative evaluation explained herein was performed at an early stage of the project in order to get feedback from end-users to improve the security activities, the associated models and the descriptions of both. The evaluation was performed by selecting a set of software developers and security experts from some of the industrial partners of SHIELDS. These were exposed to descriptions and examples of the security activities, along with scenarios that describe the context of the activities execution. The evaluation method is based on semi-structured interviews and a set of questionnaires according to Technology Acceptance Model (TAM) [4].

In the next sections we explain the SHIELDS activities and models that have been target of evaluation. We then describe the qualitative evaluation method and summarize the feedback received from industry.

This is followed by a discussion on the methodology and the results. Finally, the paper is concluded along with plans on our future work.

2 The SHIELDS activities

The SHIELDS approach is not intended to be a development process of its own, but proposes six activities that can be integrated with most existing development processes with as little extra overhead as possible.

These activities are shown as rounded rectangles in Fig. 1, where they have been related to what is considered to be the most generic phases in any development process, namely *requirements*, *design*, *implementation* and *testing*. Although the SHIELDS activities are complementary, they are not strictly dependent on each other and can be performed separately. This is similar to the seven touch-points or principles for software security proposed by McGraw [5], but our activities are always based on using security knowledge in the form of security models. We believe that this benefits access and comprehension of the information, as well as the process of sharing security information [6].

We will now briefly introduce these activities, but for a more thorough walkthrough with examples the interested reader should refer to the publicly available report on the SHIELDS Web site *D1.2 Initial SHIELDS approach guide* [7] (which was the theoretical source used during the evaluation). The modeling formalisms mentioned related to these activities are further explained in section 3.

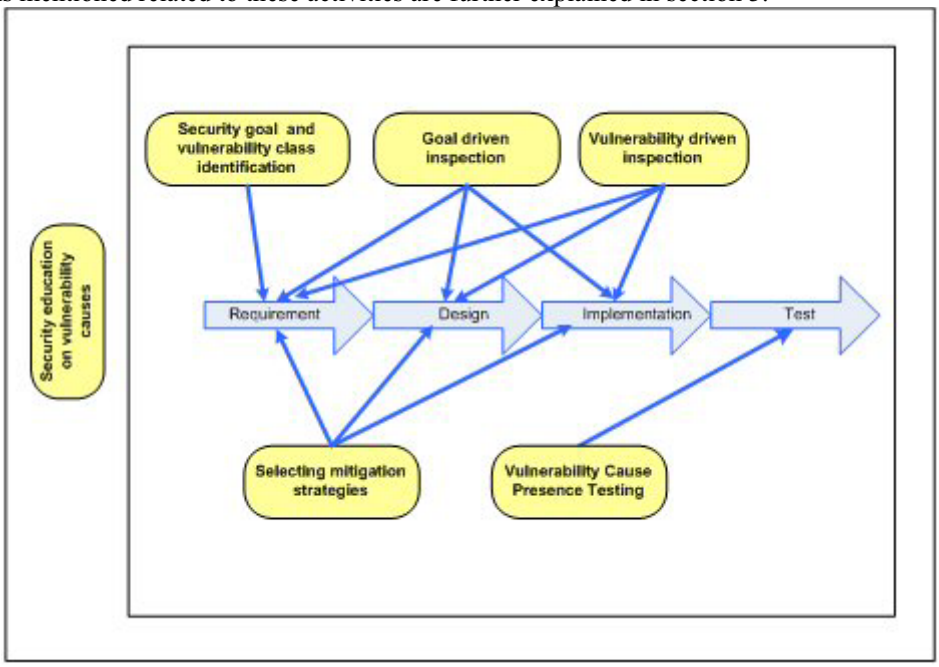


Fig. 1. SHIELDS security activities related to development phases

Security education on vulnerability causes is an activity not directly associated to any specific development phase. Developers are hardly able to design, develop and test secure systems until they understand the most common reoccurring security issues and their causes [8]. An important step toward securing software products is to raise security awareness of developers through the provision of up-to-date information about known vulnerabilities. It is important that developers learn how inadequate requirements, flaws in design, and mistakes in code can result in vulnerabilities. This can be accomplished using *Vulnerability Cause Graph* (VCG) [9], [10], which are graph structures that relate causes to vulnerabilities in a software product. Using VCGs developers can develop an in-depth understanding of vulnerabilities and their causes. This additional depth is crucial to be able to identify solutions to prevent vulnerabilities in future software products and not to repeat same old mistakes.

Security Goal and Vulnerability class identification is an activity that uses threat models to identify both the security goals for the software and the potential vulnerabilities that might occur in it. A developer creates his project specific threat models by starting from generic ones and adding details specific of the

project he is working on, or by reusing more specific models from similar development projects and adapting them to the current project needs. The threat models visually present aspects that can be of threat to a software system or a component of a system. These should be used to prioritize what is to be protected (and thus indicate security goals) and identify relevant attacks that exploit commonly found vulnerabilities for this type of system. Additionally, the models can show measures on how to mitigate the threats. The modeling formalisms we use for threat modeling are attack trees [11] and misuse cases [12].

Goal-driven inspections are manual inspections on different development documents (not just source code) that check for indicators or evidences of the correct implementation of security goals. The starting point for the inspection is a set of identified security goals. A security goal, when met, contributes to meeting some other security goal or ensures that one or more security properties desired by some stakeholder hold. Security goals are closely related to security requirements and policies, but goal-driven inspections can also be performed on the security requirements documents themselves. The technique introduces a model named *Security Goal Indicator Tree* (SGIT) [13] that describes in a tree-like structure the indicators to check for a certain security goal and their relationships. A SGIT is transformable into a *Guided security inspection checklist* [14] that consists of a set of questions for the inspector to answer during the inspection, so that inspection procedure is easily understandable and even non-security experts can perform it.

Vulnerability-driven inspections are manual inspections on different development documents that aim at searching for vulnerability causes. These inspections check for evidences in development documents that indicate that a specific vulnerability is present. In order to guide these inspections for each vulnerability class a *Vulnerability Inspection Diagram* (VID) model is used [15]. The VIDs are a high level description of the inspections and are transformable into *Security Inspection Scenarios* that explain in natural language the manual inspection procedure in even a more understandable way. Both models can be understood by non-security experts so they can perform the inspections.

Selecting mitigation strategies identifies alternative development activities that can be performed in order to prevent vulnerabilities. There are usually a number of alternatives that can be selected to address these security issues. The models that describe the different alternatives are named *Security Activity Graphs* (SAG), which show in a tree-like structure the different alternatives and their combinations. SAGs are used by developers to select the activities that best fits their corresponding development organization [16], [17].

Vulnerability cause presence testing is used to detect vulnerabilities in software products. Here, we also utilize the VCGs by formally defining the information of causes by creating *Vulnerability Detection Conditions* (VDCs) [7], which are then used through testing tools to determine whether the vulnerability is present in the final implementation.

3 The SHIELDS models and formalisms

The modeling formalisms supported by SHIELDS are both newly developed and extensions to previously existing ones. Generally, the introduced extensions have made it easier to add tool support and make the formalisms complement each other through common concepts found in the various models. New formalisms have e.g. been created to support model-based inspection at various stages of the development, something we did not find covered in existing work. Improvements have also made it possible to show how different types of models are related to each other.

In **Table 1** we present an overview of the main modeling formalisms already mentioned as the driving force of the SHIELDS activities.

Table 1. Overview of SHIELDS models and formalisms.

Modeling formalism	Purpose	Relation to other models
Misuse case	Get an overview of typical threats towards functionality commonly found in software systems, and common mitigating security activities to these threats.	Provide input for finding relevant attack trees, VCGs, SAGs, SGITs and VIDs.
Attack tree	Get an overview of how an attacker can achieve a specific attack goal, in order to	Provide input for finding relevant VCGs, SGITs

	protect a system from such attacks. Attack trees can be used to detail threats in a misuse case.	and VIDs.
Vulnerability Cause Graph (VCG)	Improve understanding of software vulnerabilities by identifying a vulnerability's causes and their relationships.	Provide Causes and paths leading to a vulnerability that will allow defining VDC. Can identify SAGs.
Security Activity Graph (SAG)	Identify software development activities that can prevent vulnerabilities by addressing their causes.	A SAG is typically associated with a cause in a VCG or mitigation of a threat in a misuse case or attack tree.
Vulnerability Detection Condition (VDC)	Describe system or application behavior in order to detect causes of vulnerabilities in the implementation and execution traces. This information is then typically used by testing tools to perform automated vulnerability detection.	Derived or part of a VCG.
Security Goal Indicator Tree (SGIT)	Describe indicators that can be examined to find if a security goal has been correctly implemented. The structured set of indicators can then be used to guide inspections.	Can be identified from misuse cases and attack trees. Can refer to Security Indicator Specialisation Trees.
Security Indicator Specialisation Tree	Give more details on an indicator that can be used for inspections, e.g. how to check for this indicator in different document types and on different platforms.	Connected to SGITs.
Guided Security Inspection Checklist	Provide an easy to use guide for how to inspect whether a security goal has been correctly implemented.	Used based on SGITs and Security Indicator Specialisation Trees.
Vulnerability Inspection Diagram (VID)	Guide inspections for a specific class of vulnerabilities.	Vulnerabilities can be identified by misuse cases and attack trees.
Security Inspection Scenario	Give concrete guidance as to how to perform the actions described in a VID in order to inspect for the vulnerability.	Identified by VIDs.

4 Evaluation method

In order to get early indications on the perceived usefulness and ease of use of the security activities and related models, we have used a scenarios-based evaluation method. The following text summarizes this method and the results from the evaluation, but for a more thorough explanation, together with the actual questionnaire forms and received answers the readers should refer to the publicly available report *D5.1. Results of First Evaluation of the Technical Work Packages* [18].

The goal of the evaluation was to get professional opinions on the following pre-defined criteria:

- **Level of usability of the activities**, i.e. learning curve, ease of use, efforts in the definition of security requirements and adaptation of existing procedures, expected model interdependencies and lack of coherence.
- **Expected impact** on security and trust of the software produced when performing the activities.
- **Return on Investment (ROI)** that can be expected when adopting the activities in software development processes: expected gain in efficiency, productivity and costs.
- **Potential scalability problems** when used in more complex systems compared to the provided examples.
- **Possibilities of reusing work** (models) from other projects or other users.
- **Possibilities related to adaptations and extensions** to SHIELDS.
- **Compliance with existing development processes** currently used within the organization.

The subjects of the evaluation were provided with two documents that described the descriptions and examples of the activities and models [7] and a set of scenarios showing the larger context of their use [19]. More documents from the SHIELDS project were also made available in case the subjects felt that they needed more details on the technical background of what they were evaluating, but these were not mandatory reading. Additionally, a briefing was made for the participants to explain the expectations from the evaluation and also to present some scenario walkthroughs where example models were showed. During the evaluation, human guidance was also available to assist and clarify any unclear parts of the documents.

The characteristic of the people we wanted for this evaluation were the following:

- Knowledgeable persons from industrial end-users in SHIELDS consortium not having participated in creating what was to be evaluated.
- Security experts familiar with security best practices and tools, as well as practical security reviews, threat analysis and preferably security modeling.
- Experienced software developers that are somewhat knowledgeable of current "best practices" related to secure development.
- People with research experience within the field of software security, with good knowledge of security-related sources of information (such as NVD stats, CERT stats) and various threat level measurers.

Based on this we selected four participants, two from each participating organization. The first organization (*A*) was an SME, while the second organization (*B*) was a larger enterprise. The two organizations are located in different European countries working on somewhat different types of development projects, but both concerned about software security. From organization *A* two participants fit within the description of "Software security expert and researchers (with good knowledge of various security vulnerabilities and an interest in security techniques and models)". From organization *B* the two remaining were characterized as "Software developers (involved in all phases of software development, from specification to maintenance)".

The feedback from the participants was collected using a questionnaire based on the *Technology Acceptance Model* (TAM) [4], followed by an interview performed during a one day session meeting. The interview guide consisted of detailed questions on particular scenarios, questions related to business indicators and improving the SHIELDS activities and models. The questionnaire was tried out beforehand during a pre-test on an independent security expert in order to make sure that they did not lead to any misunderstandings. The interviews were also practiced on beforehand, and performed by two different interviewers. Also four people were involved in the evaluation to analyze the results and draw conclusions.

5 The feedback from the industry

In general, the evaluators (also called subjects of evaluation) found that the security activities and models developed so far have a great potential of usability. To quote one of the subjects; *"SHIELDS technology can be applied in different fields where high system reliability is needed (e.g. industry, telecommunication or medical fields)"*. It was also expressed that the techniques will be useful in *"all development phases, from system design to system implementation, test and monitoring"*.

Nevertheless, subjects also felt that not enough material was given to evaluate the methods completely, and although they agreed that activities will help in the detection and avoidance of vulnerabilities, they did not see clearly if it will be easier to eliminate vulnerabilities. An important point is that they hesitated to state that they would actually use the activities unless it was already an integrated part of their daily development process. In subjects' opinion, the activities should probably be performed by a security expert in the design phase, by a software developer in the implementation phase and by both a security expert and software developer in the testing phase.

To adequately support different types of users it should be possible to easily locate the relevant information (models) that could be used for their particular tasks. Eventually, it should be possible to limit the models and information that is available for a given type of user. The profusion of different model types could make things more difficult to understand for the users. One evaluator suggested that *"it should be possible to say that the more popular development processes are covered by SHIELDS"*.

Related to scalability, it was stated that based on the current documentation it was difficult to see how the activities would perform in complex situations. It was recommended that the SHIELDS activities

should be designed so that complex and critical systems can be targeted and more complex examples are presented when describing the activities in the documentation.

As for possibilities for reusing work (models) from other projects or other users, subjects believed that it would be possible to use a centralized database of known security problems and models in future projects. The results of each project can be considered as part of the information/models that the user has to collect to build a complete security database. Additionally, SHIELDS can take profit from the existing major security projects and vulnerability databases. Vice versa, a public API could be created to let other systems use the centralized repository as input. Evaluators also pointed out that statistics should be used to help users find the models and techniques in the repository that are most popular to solve problems that are similar to their own. The effort needed to feed repository with usage statistics should be reduced as much as possible for users or they will be reluctant to provide them.

Concerning the perceived ROI of adopting the activities, evaluators asserted that the prevention of expensive security flaws like loss of data and leak of sensitive information was an important aspect that would make the adoption of the SHIELDS activities profitable and would allow selling better value to customers. About the costs of development, software developers will gain in efficiency and productivity since they can easily find the relevant security information they need in a short period of time. Using the activities should make it faster and simpler to do validation and testing. Product maintenance should also become cheaper, thus improving customer satisfaction.

One of the crucial factors that were pointed out is the necessity to complement the activities with automation tools (at least during implementation and testing). The evaluators believed that special attention should be given to building easy-to-use interfaces for the users and well-defined API's for integrating new tools. One evaluator recommended supporting automatic and periodically analysis of software products to detect whether any new vulnerability is introduced (when modifying the product) or to take into account the new vulnerabilities information added to the centralised repository.

Other suggestions were in the line of improving the description and/or content of some of the usage scenarios or the documentation itself, mostly with examples that show the reusability of the models in other projects and how external vulnerability information databases and security tools can be integrated in the approach.

6 Discussion

There are a number of advantages of having performed a scenario-based evaluation early in this research project, e.g. it helped to improve usability and to eliminate misconceptions and lack of completeness in the activities and modeling formalisms we are working with. Another important issue has been to verify that the scenarios describing the use and context of the activities and models are realistic and achievable in a real-world setting. The results have given us many indications on how to proceed, such as the need for more complex examples when presenting the models and activities. Simple school-book examples are good for basic understanding, but in our case we need more realistic and detailed ones to show the benefits of adopting the SHIELDS activities in development of complex systems.

The two major factors that reduce the significance of this qualitative evaluation are:

1. The evaluation was mainly based on documentation of the SHIELDS activities and models, but no real trial on using the models or performing the activities during the development of a real application was carried out.

2. Only four people from two software companies participated, which is hardly a number that provides substantial evidence.

Regarding the first factor, we chose to do it this way in order to introduce the end users to the activities and models as early as possible. At the time of the evaluation, the supporting tools were still very immature, which would probably have stolen a lot of the attention of the evaluators. For the next evaluation the evaluators will make a more hands-on test of the SHIELDS activities, models and supporting tools. This will give them a better idea of which aspects of their work will be impacted and how, and we expect to obtain richer feedback.

As for the second factor, the number is low, but we believe that we selected representative candidates, and involving more people would probably not have given us much more fruitful feedback. We saw from the results that the evaluators were pretty much in agreement, which supports this assumption. The next

evaluation will of course involve more people so that we can support our work with more evidence and measurements.

7 Conclusion and future work

The qualitative evaluation presented herein was performed during the first ten months of the SHIELDS project (with a total duration of 30 months) on descriptive documentation of the model-based security activities, examples and usage. As a general conclusion from the evaluation, it is believed that adopting these activities will bring benefits to the current software development processes used by the software companies, helping developers to implement reliable software and eliminate vulnerabilities in their products. There is a great interest in easy and efficient solutions to guide developers during their tasks (i.e. conception, implementation and testing) to improve the software security. The interest will be significantly improved if the activities are supported by automation tools (especially during implementation and testing), something which is a major goal of the project, but was not ready for this evaluation.

The next evaluations within SHIELDS are planned for the end of phase 2 (June 2009) and end of the final phase 3 (June 2010). These will be both qualitative and quantitative evaluations aimed at assessing the usefulness and easy of use when actually performing the activities, including the practical appliance of models both manually and through supporting tools. We will also evaluate creation/modifying models through the use of the centralized model repository.

Acknowledgements

The research leading to these results has received funding from the European Community Seventh Framework Programme (FP7/2007-2013) under grant agreement no 215995. We would also like to acknowledge all the members of the SHIELDS Consortium for their valuable help. Especially we would like to thank Professor Nahid Shahmehri and David Byers from Department of Computer and Information Science at Linköping University for their excellent work in the SHIELDS modelling formalisms and security activities.

References

1. Davis, N.: Developing Secure Software, Software Tech News, vol 8, nr 2, 2005.
2. SHIELDS Project Consortium: SHIELDS Project Homepage, http://www.shields-project.eu
3. Meland, P.H., Ardi, S., Jensen, J., Rios, E., Sanchez, T., Shahmehri, N., Tøndel, I.A.: An architectural foundation for security model sharing and reuse, Proceedings of the Third International Workshop on Secure Software Engineering (SecSE), IEEE Computer Society, Fukuoka, Japan, March 2009.
4. Davis, F.D.: Perceived usefulness, perceived ease of use and user acceptance of information technology, MIS Quarterly 13 (1989) 319–340.
5. McGraw, G.: Software Security: Building Security In, Addison-Wesley, 2006.
6. Ardi, S., Byers, D., Meland, P.H., Tøndel, I.A., Shahmehri, N.: How can the developer benefit from security modeling?, Proceedings of the Second International Conference on Availability, Reliability and Security, ARES2007, IEEE Computer Society, pp. 1017-1025, Vienna, Austria, April 2007.
7. SHIELDS Project: D1.2 Initial SHIELDS approach guide. Report 2009. http://www.shields-project.eu
8. Howard, M.: Building more secure software with improved development process. IEEE Security & Privacy, 2(6):63–65, 2004.
9. Ardi, S., Byers, D., Shahmehri, N.: Towards a structured unified process for software security, Proceedings of the ICSE 2006 workshop on Software Engineering for Secure Systems (SESS06), Shanghai, China, 2006.
10. Byers, D., Ardi, S., Shahmehri, N., Duma, C.: Modeling software vulnerabilities with vulnerability cause graphs, Proceedings of the International Conference on Software Maintenance (ICSM06), Philadelphia, USA, September 2006.
11. Schneier, B.: Attack Trees, Dr. Dobbs Journal, December 1999.

12. Sindre, G., Firesmith, D., Opdahl, A. L.: A reuse-based approach to determining security requirements. In Proceedings of the 9th international workshop on requirements engineering: foundation for software quality (REFSQ'03), Klagenfurt, Austria, 2003.
13. Peine, H., Jawurek, M., Mandel, S.: Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection. HASE 2008: 9-18
14. Elberzhager, F., Klaus, A., Jawurek, M.: Software Inspections Using Guided Checklists to Ensure Security Goals, Workshop on Secure Software Engineering, Fukuoka, Japan, 2009.
15. SHIELDS Project: D4.1. Initial specifications of the security methods and tools. Report 2009. http://www.shields-project.eu
16. Byers, D., Shahmehri, N.: Prioritisation and Selection of Software Security Activities, Fourth International Conference on Availability, Reliability and Security, ARES 2009 (IEEE Computer Society ed.), Fukuoka, Japan, March 2009.
17. Byers, D., Shahmehri, N.: A cause-based approach to preventing software vulnerabilities, Proceedings of the International Conference on Availability, Reliability and Security (ARES08), Barcelona, Spain, March 2008.
18. SHIELDS Project: D5.1. Results of First Evaluation of the Technical Work Packages. Report 2009. http://www.shields-project.eu
19. SHIELDS Project: D1.1. Initial architecture and requirements specification. Report 2009. http://www.shields-project.eu

Toward model-based security engineering: developing a security analysis DSML

Véronique Normand[1] and Edith Félix[2]

[1]Thales Research & Technology [2]Theresis
1 avenue Augustin Fresnel
91767 Palaiseau cedex
FRANCE

{veronique.normand, edith.felix}@thalesgroup.com

Abstract. As a long-term industrial initiative, we are developing a new method to support security risk analysis, closely integrated with the overall engineering process of our critical information systems. This method is building upon model-based engineering techniques. This paper presents a prototype domain-specific modelling language (DSML) that was developed in this context; this DSML aims at supporting the analysis and assessment of security risks for a system, and the specification of requirements for security measures to address those risks. The rationale, approach and main features of this DSML are presented. Our objective is to provide adequate and efficient tooling to security engineers for an effective integration of security engineering in the process of critical system design, so as to enable a better targeting of security specifications.

Keywords: MDE, DSLs, Security, Risk analysis

Introduction

Critical systems engineering involves a complex multi-disciplinary set of processes and teams to define, develop or acquire, verify, integrate and validate these systems. A particular emphasis is laid on non functional requirements, including safety, dependability, performance and security. These non functional requirements are key drivers to the architectural design of the system; they thus need to be addressed early in the lifecycle, in close integration with the functional and architectural design of the system.

This is putting requirements for systems engineering methods and tools to support the capture, articulation, trade-off and reconciliation between multiple viewpoints [0] over a system architectural design. This paper focuses on system security engineering as a particular modelling viewpoint [0] in an overall system modelling framework.

System security engineering is defined as the effort to achieve and maintain optimal security and survivability of a system [0]. This involves security risk analysis, specification of security requirements, and design of security means over the system architecture. We are addressing the enhancement of traditional security risk analysis methodologies based on modelling techniques that will allow leveraging detailed knowledge of the targeted system in close integration with the mainstream system engineering process, and developing fine grain analyses of the actual risks at stake.

This paper presents a prototype domain-specific modelling language (DSML) that was developed in this context; this DSML aims at supporting the analysis and assessment of security risks for a system, and the specification of requirements for security measures to address those risks. It is organised as follows:

- Section 2 provides the rationale for this work, and presents the principles of our approach to enhancing classical security analysis methods.
- Section 3 presents the DSML: the approach taken to developing this DSML, the domain covered, the actual contents of the DSML (extract of metamodel and graphical syntax), and a quick comparison to the state of the art.
- Section 4 concludes with status information and perspectives regarding our security engineering research.

Context and rationale

System security engineering classically involves: 1) the analysis and assessment of security risks encountered by the system, 2) the specification of requirements for security measures to address those risks, and 3) the design, development, integration and validation of a security architecture, functions and mechanisms that address those requirements.

Our present work is focusing on security engineering activities 1 and 2 above. Our objective is to provide adequate and efficient tooling to security engineers for an effective integration of security engineering in the process of critical system design; this will enable a better targeting of security specifications.

Enhancing system security engineering

A comprehensive approach of security engineering starts with an analysis of the risks pending on the system. For critical systems, this analysis must be conducted with the biggest attention since the impacts can be very damaging for the populations in relation to those critical systems. On the other hand, overestimating risks may lead to excessive or unnecessary security measures, introducing undue rigidities and costs. The specification of security requirements builds upon this risk analysis, and aims at defining requirements that are commensurate with the risks.

Currently in our company, Security Analysis activities are carried out with the help of structured and proven methods that use referential repositories (of types of threats and vulnerabilities, of impacts and damages, attack scenarios, security functions etc.), standardized or not, and tabular, cross-matrix and dashboard based tools. EBIOS [0] and MEHARI [0] are the main methods employed at our company.

These methods imply a limited perception of the architecture of the system upon which the risk analysis is realised. In particular, we carried out a study of these methods and realized that they target on the one hand the business process supported by a system and on the other hand, in very little detail, technical and physical elements of the system (applications, databases, data files, servers, networks, mobile PCs etc.). Finer-grain knowledge of the architecture is not taken into account in these methods. The topology, data flows and functional dependencies throughout the system are especially not analyzed, which can lead to sub-optimal risk analyses and security requirements specifications.

Our work aims at developing a method that enables an enhancement of these classical risk analysis methodologies. As summarised in Figure 1, these enhancements rely on leveraging detailed knowledge of the targeted system in close integration with the mainstream system engineering process, and developing fine grain analyses of the actual risks at stake. This method builds upon the capacities provided by model-based development methods and techniques that are currently spreading in the systems engineering community, but are still poorly used in the security engineering domain.

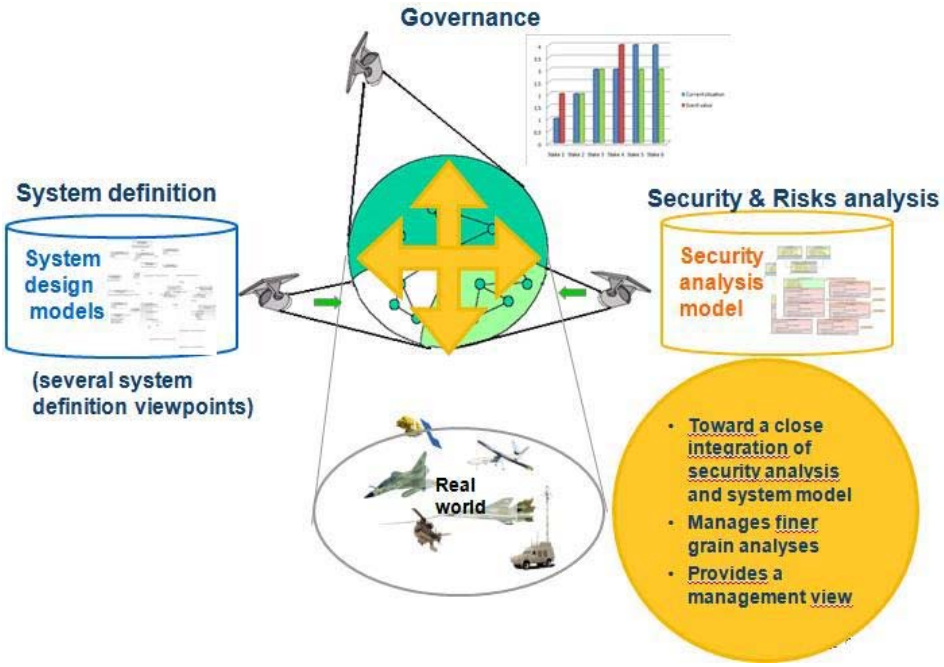


Figure 1: Enhancing system security engineering methods

Our general objectives of enhancement are the following:

- *Objective1:* To optimize the qualification of the risks and the specification of security requirements and related security costs,
- *Objective 2:* To optimize the quality and the productivity of security engineering by capitalizing on data from one study to the next, and by proceeding to automatic calculation and consistency checking.
- *Objective 3:* To optimize the quality and the productivity of security engineering by sharing common models of the system between system design and security analysis and thus by working on synchronized and consistent models of the system throughout the design process.

The security risk analysis method: principles

Our prospective security risk analysis method builds upon model-based engineering methods and techniques. All activities of our method are organised around the building and usage of models, that is formalised, precisely defined, interconnected and integrated representations of the objects under study.

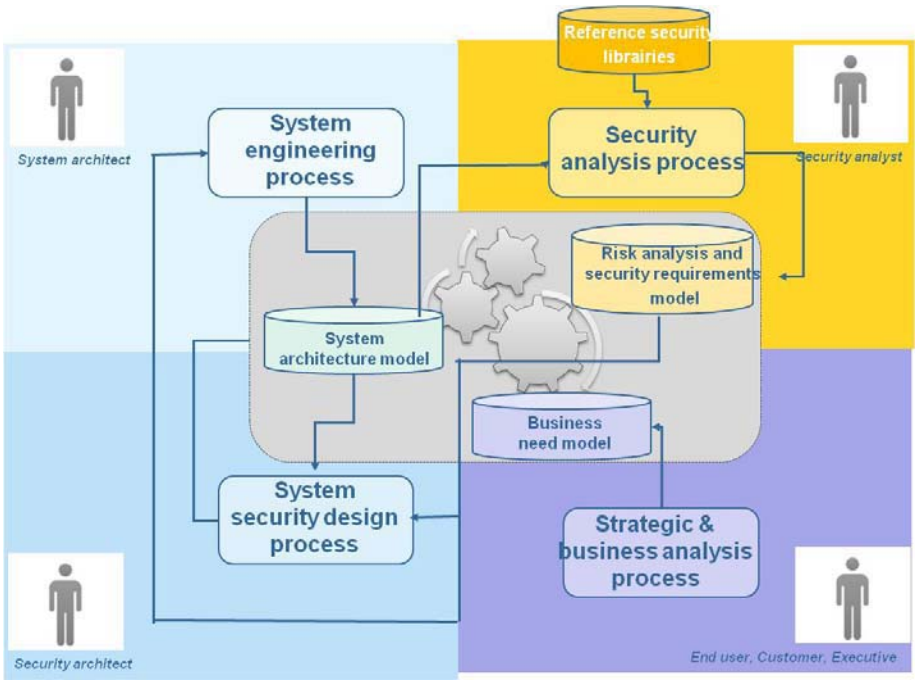


Figure 2: The security analysis method in context – big picture

As represented in Figure 2, our proposed method relies on the development of a modelling framework that combines in a synchronised way a set of models that constitute separate viewpoints [0] over the engineering problem:

- The System architecture model contains the architectural design of the system; this model is developed within the mainstream engineering processes, along at least two dimensions: the functional / logical architecture of the system (functional capacities and data to be realised by the system) and the physical /implementation architecture of the system (actual hardware and software components that realise the functional capacities).
- The Business need model captures a representation of the business context for the system: business process that is supported, underlying business organisation, business objects, key performance indicators, strategic drivers, etc.
- The Risk analysis and security requirements model captures the results of the security risk analysis method that is proposed here. This model includes a representation of the system architecture that is relevant to the needs of the security analyst; this representation is traced back and maintained in synchronisation with the system architecture model (see section 4). The security risk analysis information is defined as annotations or related new concepts added over the system architecture elements. The risk analysis and security requirements model may also be traced to elements of information defined in the Business need model.

The System architecture model and the Business need model are part of a systems architectural modelling framework that we are developing to address service-oriented types of large-scale enterprise integration systems or systems of systems. This modelling framework leverages the MODAF architecture framework [0] and the ongoing SOAML [0] standardisation effort, and extends these in two directions: 1) a structuring capability analysis pattern is developed at the core of the framework; 2) technical software integration architecture is addressed, targeting a specific technical platform. This framework is described in [0].

The Risk analysis and security requirements model is expressed in a dedicated DSML. This DSML is presented in section 3 hereafter.

The prototype Security DSML

Approach for developing the DSML

The development of our security analysis DSML was carried out via a close collaboration between MDE experts and Security domain experts. An integrated team was set up, involving:

- A person responsible for the development of the DSML (called the "DSML developer"). This person is experimented in DSML technical development, with a good expertise in the Eclipse GEF/GMF environment [0], which was retained for this work.
- Modelling experts, with a good expertise in meta-modelling and language engineering (but limited knowledge of the Eclipse GEF/GMF environment).
- A security expert experimented in security risk analyses, with a good knowledge of security methods, and a keen interest in improving these methods, and with sufficient availability to support our work on a daily basis.
- A number of other security experts with more limited availability. Meetings were organised with these experts so as to collect and validate requirements and priorities.

The development of the DSML was an iterative process that involved the following activities:

- Scoping the DSML: a pre-study was conducted with the security experts to analyse security practices, identify core areas of interest, and determine the target perimeter of the DSML. This work involved a state of the art analysis that covered industrial security risk analysis methodologies (mainly EBIOS and MEHARI), but also academic work in this area.
- Meta-modelling the domain: the security analysis meta-model was developed in close collaboration between security and modelling experts. The analysis of EBIOS data (as described in the documentation of this method) provided an initial, high-level understanding of the domain. Detailed domain knowledge was then elicited and refined into meta-models through a number of techniques involving active listening, re-phrasing, the production, explanation and validation of abstract, conceptual models. This allowed to gradually gather a shared understanding of the concepts of interest and the nature of relationships between them. A constant concern was also to refine the perimeter of interest and determine which concepts were to be included and which ones left aside from our Security DSML.
- The production of the "technical" meta-model was then realised. This technical-level meta-model was not directly discussed with the domain experts as it is less easy to understand than conceptual models.
- The design of the graphical syntax was realised together with the domain experts.

The iterative process bears similarities with rapid prototyping practices that are common in human-interface design situations: several versions of the DSML prototype were gradually developed and validated with the security expert through hands-on sessions.

Domain of interest

Our goal is to build a DSML allowing the support of finer grain, more formal security analyses that exploit formalized system architecture descriptions. The security architect formalizes security information and relates it to architecture components.

Figure 3 illustrates the scope and context of use for our "Security DSML":

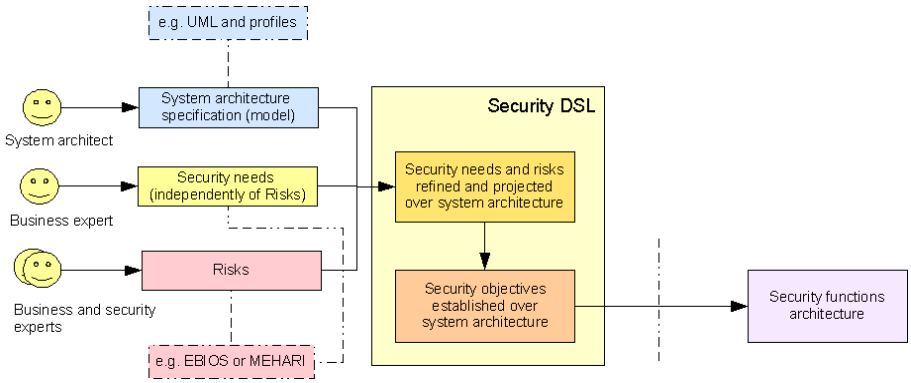


Figure 3: Scope of the Security DSML

- The System architecture model is built using distinct languages (like, for example, UML and/or UML profiles) by a System architect. *The Security DSML is defined with limited dependency upon the specific system architecture description formalism*. The objective is to be able to use the DSML concurrently with different languages for the specification of system architectures.
- Security needs are determined for each individual architecture components or groups of such, by a Business expert. A Security need is initially expressed intrinsically (e.g. "The document needs to be defence confidential"), without taking into account the risks, but only the impacts of unwanted actions and damages they may inflict.
- A Risk Analysis takes place, involving the collaboration of the Business and Security experts, in order to identify and value risks regarding system components and subcomponents.

The Security DSML shall support security needs and risks to be refined and projected on a System architecture model.

The DSML shall then support the experts work in determining which risks are unacceptable towards the specified security needs, either because they have a too important impact, a too big opportunity of happening, or both, making these components critical.

Security objectives are defined in order to reduce unacceptable risks and consequently bring the current level of security to a newly defined targetted one. The Security DSML shall support the capture of these security objectives and the assessment of their coverage of unacceptable risks pending on architectural components.

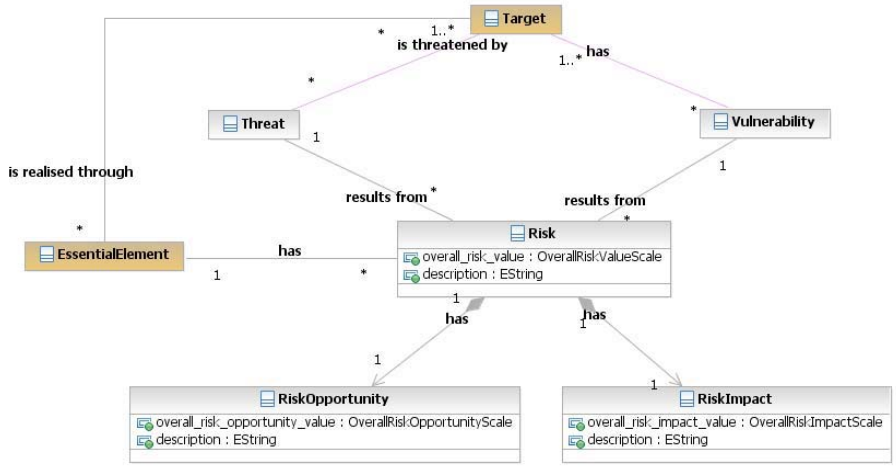
The actual definition of a security architecture satisfying these objectives is outside the scope of this DSML.

Metamodel

This paper cannot be the place for a detailed presentation of the metamodel and syntax of our DSML. We are providing below representative extracts. More details are provided in [0]. The core part of the metamodel[2] is represented in Figure 4 below.

The system under analysis is considered to hold targets and essential elements. *Targets* are physical elements subject to vulnerabilities and damages by threats. *Essential elements* are usually more logical, functional elements: data and functions (or services, or capabilities depending on context) that are essential to the business stakes of the company, and therefore subject to security needs. Essentiel elements depend on targets for their implementation.

[2] For readability sake, it is represented in the form of a conceptual model rather than a formal metamodel.



- **Figure 4: The Risk related meta-model view – conceptual model**

The central concept of our security analysis metamodel is the one of *risk*. A risk pertains to an essential element of the system. A risk comes from the combination of a *threat* that could exploit an opportunity to take advantage of a *vulnerability* of a target, with the *essential element* depending on the *target*. A risk is valuated based on its impact on the target and its opportunity to be triggered on the target.

The link between the risk analysis model and the system architecture model is illustrated in Figure 5.

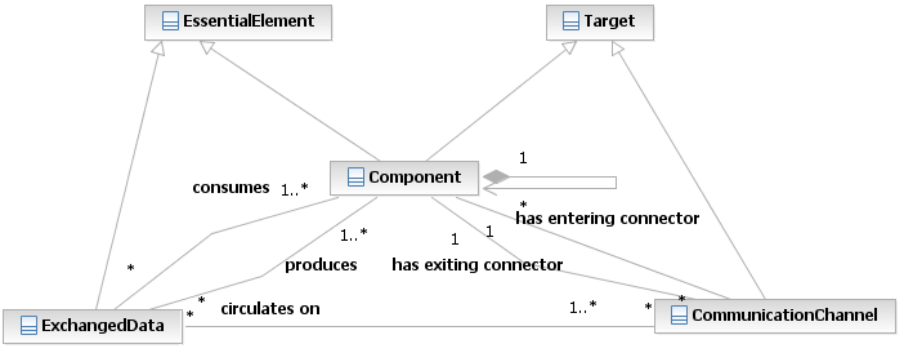


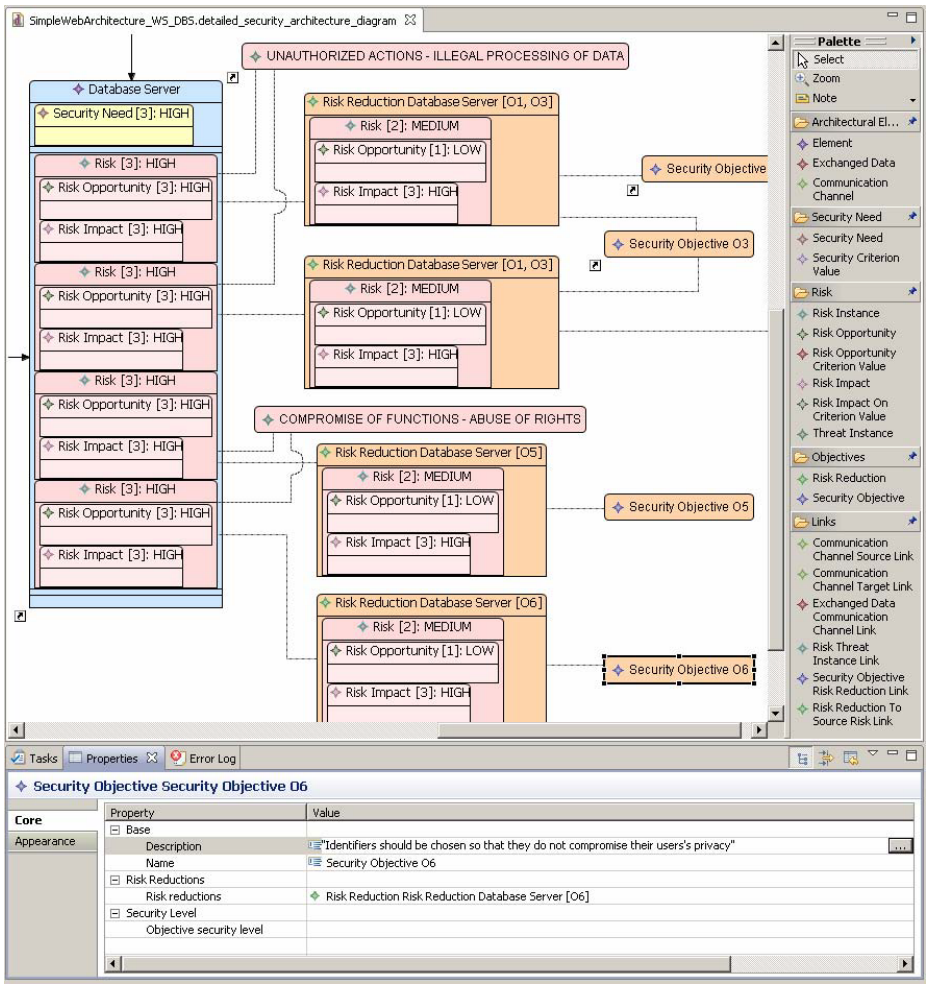
Figure 5: Linking architecture description to risk analysis – conceptual model

Our approach was to introduce in the DSML placeholders for system architecture elements generic enough to support mapping to various types of architectures (whether business, system or technical). In our current prototype DSML, only three generic architectural concepts are used, as represented in the figure: components, communication channels linking the components and data exchanged between components via the communication channels. Data correspond to essential elements while communication channels are considered as targets; components are considered as both essential elements and targets. This simplified model bears a number of limitations and is an area for future work.

Graphical syntax and modeller tool

A prototype modeller was developed using Eclipse GEF / GMF.

The graphical syntax was designed as part of a quick prototyping work. An illustration of this syntax is provided in Figure 6, which shows an example set of diagrams that use the DSML.


SimpleWebArchitecture_WS_DBS.detailed_security_architecture_diagram
UNAUTHORIZED ACTIONS - ILLEGAL PROCESSING OF DATA
Database Server
Security Need [3]: HIGH
Risk [3]: HIGH
Risk Opportunity [3]: HIGH
Risk Impact [3]: HIGH
Risk Reduction Database Server [O1, O3]
Risk [2]: MEDIUM
Risk Opportunity [1]: LOW
Risk Impact [3]: HIGH
Security Objective
Security Objective O3
COMPROMISE OF FUNCTIONS - ABUSE OF RIGHTS
Risk Reduction Database Server [O5]
Security Objective O5
Risk Reduction Database Server [O6]
Security Objective O6
Palette
Select
Zoom
Note
Architectural El...
Element
Exchanged Data
Communication Channel
Security Need
Security Need
Security Criterion Value
Risk
Risk Instance
Risk Opportunity
Risk Opportunity Criterion Value
Risk Impact
Risk Impact On Criterion Value
Threat Instance
Objectives
Risk Reduction
Security Objective
Links
Communication Channel Source Link
Communication Channel Target Link
Exchanged Data Communication Channel Link
Risk Threat Instance Link
Security Objective Risk Reduction Link
Risk Reduction To Source Risk Link
Tasks
Properties
Error Log
Security Objective Security Objective O6
Core
Appearance
Property
Value
Base
Description
"Identifiers should be chosen so that they do not compromise their users's privacy"
Name
Security Objective O6
Risk Reductions
Risk reductions
Risk Reduction Risk Reduction Database Server [O6]
Security Level
Objective security level

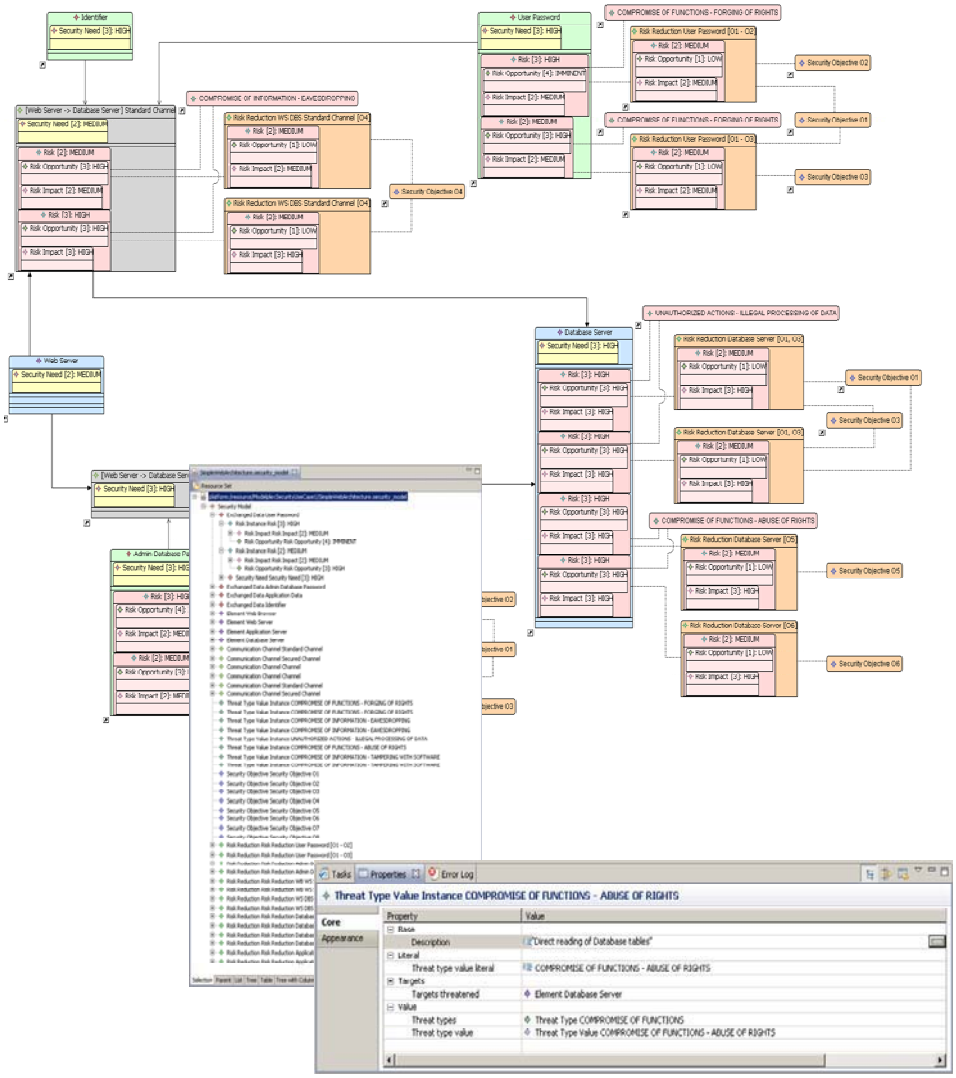


Figure 6: Example diagrams using the Security DSML

Comparison to existing work

The research community in the security analysis area tends to focus on specific techniques supporting aspects of the risk analysis work, such as the modelling of attack scenarios, vulnerability cause graphs, use and misuse cases, attack trees, etc. These specific techniques are complementary to our work: they can offer detailed modelling support that could possibly be integrated in our approach.

Very few research initiatives address the security assessment problem as a whole. The CORAS project [0] is to our knowledge the main piece of work in this area. CORAS is addressing a scope that is similar to the one we are addressing, but serves different objectives: CORAS is aiming at supporting brainstorm sessions between security analysis stakeholders, while our DSML addresses fine-grain security analyses over a system architecture. The CORAS approach and its supporting information model are thus significantly different from ours. CORAS does not investigate the integration of the security risk analysis process with the system engineering process; the approach does not exploit at fine grain the system architecture information.

Status and perspectives

This paper has presented a prototype DSML for security risk analysis. This DSML was developed as part of a first iteration of work, in the context of a longer-term research work that aims at developing an enhanced model-based method for the security engineering of critical information systems in our company. This work allowed to achieve a proof-of-concept prototype, with a focus on scoping and capturing a relevant meta-model rather than on developing high-quality diagrammatic notations and tooling. In particular, the prototype DSML in its current state clearly bears shortcomings regarding ergonomics and usability.

The proof-of-concept objective was achieved; sufficient interest was raised within our company so as to support the continuation of this initiative. Directions for this work are multiple:

- Enhancing the security analysis DSML in several areas, where simplifications were made in our initial prototype. For example, a refinement of the stakes / needs / damages model is being considered, which would support a more precise computation of risk severity.
- Including automated computation formula and consistency checking rules.
- Integration of the DSML with our system modelling framework is another area of work. This work is taking place within the frame of a more general research on the support to multi-disciplinary engineering, where technical approaches to heterogeneous modelling viewpoint integration are being developed.
- Finally, future work includes complementing our risk analysis DSML with modelling and tools for supporting security solutions design and verification, thus extending our scope to fully address our model-based security engineering target.

Work has begun along these perspectives; this work will continue in the context of newly launched FP7 project SecureChange.

Part of our work will address a comprehensive validation of our proposed approach against the objectives described in section 2.1, as new, enhanced versions of the DSML are produced. While analytical, criteria-based quality assessments can be of interest, our feeling is that an empirical method is needed here. Our evaluation method will first focus on assessing how the tooled DSML achieves objective 1 (refer to section 2.1), that is the objective of optimizing the qualification of the risks and the specification of security requirements and related security costs. We are considering an approach where two experientation campaigns are realised in order to compare security analysis performed with both the DMSL tooling and with a classical tooling such as the EBIOS software:

- In order to evaluate the gain in terms of quality or accuracy of the measurement of risks and of the security level of security requirements and the associated costs, a security risk analysis would be performed on a given system by the same experts alternatively with both type of tooling.
- In order to evaluate the gain in time, two risk analysis on two different systems having the same complexity could be performed by the same expert, one analysis with the DMSL tooling, the other analysis with a classical tooling.

Assessment of objective 2 (optimization of the quality and the productivity by capitalisation and automatic calculation and consistency checking) may be more difficult. The automatic calculation part could be assessed as part of the experimentations described above. Assessment of the capitalisation part needs to be investigated, probably in long-term operational usage contexts.

Objective 3, which relates to the benefits retained from more integrated collaboration between systems engineering teams and security experts through the sharing of models, is more complex to assess. Pilots in operational contexts may be the most realistic option here.

Acknowledgment. The work reported in this paper was supported by the European Commission via the MODELPLEX project, co-funded under the “Information Society Technologies” Sixth Framework Programme (2006-2009).

References

IEEE Architecture Working Group. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000, IEEE. (2000).

National Information Systems Security (INFOSEC) Glossary, NSTISSI No. 4009, Revision 1. (1999).

Jouenne, E. and Normand, V. Tailoring IEEE 1471 for MDE support, in UML Satellite Activities (N. J. Nunes, B. Selic, A. R. da Silva, and J. A. T. Álvarez, eds.), vol. 3297 of Lecture Notes in Computer Science, pp. 163–174, Springer. (2005).

MEHARI method, https://www.clusif.asso.fr/fr/production/ouvrages/type.asp?id=METHODES

EBIOS method, http://www.ssi.gouv.fr/en/confidence/ebiospresentation.html

Jouenne, E. et al.: Systems of systems modelling framework. MODELPLEX project restricted deliverable 2.2.b. (2008).

MODAF, http://www.modaf.org.uk

OMG, Service oriented architecture Modeling Language (SoaML) – Specification for the UML Profile and Metamodel for Services (UPMS), OMG document ad/2008-08-04. (2008).

Eclipse GEF/GMF, http://www.eclipse.org/modeling/emf/, http://www.eclipse.org/gmf/

Normand, V., Felix, E., Jitia, C. A DSML for security analysis, IST MODELPLEX project restricted deliverable 3.3.g. (2009).

Lund, M., den Braber, F., Stølen, K., Vraalsen, F. : *UML profile for the identification and analysis of security risks during structured brainstorming*. Technical report STF40 A03067, SINTEF ICT, (2004).

From Security Modelling to Run-time Security Monitoring

Antti Evesti, Eila Ovaska and Reijo Savola

VTT Technical Research Centre of Finland, Kaitoväylä 1,
90571 Oulu, Finland
{Antti.Evesti, Eila.Ovaska, Reijo.Savola}@vtt.fi

Abstract. In this paper we take the first steps from security modelling to run-time security monitoring. Providing full support for run-time security monitoring requires that following issues are solved: security concepts has to be defined in an unambiguous way, security level has to be defined and measured, and finally, software has to adapt itself based on measurements and requirements. This paper addresses the unambiguous definition of security by examining existing security ontologies. None of the existing ontologies is able to support run-time security monitoring as such, and there is a need to combine and widen these ontologies. In addition, this paper describes our vision how run-time security management can be achieved as the wholeness.

Keywords: Security ontology, security measuring

1 Introduction

Today's software products are not running in a static and beforehand known environment. Instead, software is running in mobile devices within constantly evolving environments or, alternatively, software is running in a fixed place but new services become available for exploitation of this device. In addition, the threat landscape is changing constantly. In both cases, the user wants to preserve a particular security level (or security performance) – even though his/her environment changes. In these circumstances, making all security decisions at a design time is not sufficient and thus managing security also at run-time is required. In other words, it is necessary to reveal security changes, by means of monitoring, and adapt the software during its execution – in order to ensure the desired security level for system's users.

Although there are a number of security solutions introduced in the literature [10], [22], there is no common understanding how to define and measure security in practical cases. The existing definitions are generic but too abstract. One example of immaturity of methods and techniques used for security modelling and measurement is the diversity of security concepts and metrics defined in research papers and standards, for example in [1] and [3]. Moreover, security is a cross-cutting issue [2], and therefore, its management is difficult, not only at run-time but also at design-time.

Development of appropriate run-time security solutions is a complex process. Firstly, security has to be defined, i.e. modelled, in an unambiguous way that is universally understood by all stakeholders. These generic security models, typically represented by means of security ontologies, are used as basis of security aware software development. Secondly, an appropriate security level needs to be defined, measured and monitored constantly. Obviously, a widely-accepted measurement ontology is needed to enable metrics development. Thirdly, software has to be able to adapt itself based on the measurement results. In this paper, we concentrate on the first challenge, security modelling, by examining security ontologies that can be used for representing security. Since these ontologies offer support for security measurement, security measurement issues are also discussed. Finally, we introduce our vision about the run-time security management based on existing security ontologies that are to be combined and enhanced.

The remainder of this paper is organized as follows. Firstly, existing security ontologies are examined, and thereafter, quality and security measurement issues are discussed. Section 3 shows how our approach takes security issues into account at design-time, and, thereafter, we present our vision of the run-time security management. Conclusions and future work section closes the paper.

2 Security Ontologies

2.1 Information Security in General

In order to achieve coherent understanding of security we have to describe security issues in a universal way and incorporate enough details to make the description useful. Ontologies make it possible to give this kind of presentation. We used the following method to select ontologies to this study: 1) ontologies have to concentrate information security, 2) only general purpose ontologies, i.e. not domain specific ontologies, and 3) ontologies have to be mature enough, i.e. at least a concept structure has to be available. Based on these criteria, we describe four existing security ontologies and consider their differences – concentrating applicability to run-time usage and security measurement.

Savolainen et al. present a taxonomy of information security for service centric systems in [7]. The presented taxonomy is intended for the use of software architects of service centric systems. Thus, the taxonomy supports following aspects: 1) stakeholders' participation in the development of service-centric systems, 2) improve communication of security concerns in requirements elicitation, 3) aid to designing and constructing of security means while architecting and 4) support quality analysis phase by providing a common security terminology. [7]

The security taxonomy is divided to five main concepts as shown in **Fig. 7**. *SecurityAssets* contains entities that have a value, i.e. asset means an entity that has to be protected. *SecurityAttributes* contains concepts *Confidentiality*, *Integrity* and *Availability* – also called CIA triad. It must be noted that in telecommunications, this triad is often enhanced with *non-repudiation* and explicit reference to *authentication* and *authorization*. These security attributes compose service's security, and thus, security specification of a system should cover all of these aspects. After that, the concept *SecurityThreats* defines *Faults*, *Errors* and *Failures*. A failure is defined as an event that occurs when the delivered service differ from the correct service. Instead, deviation in the external state of the system is called an error and the cause of an error is called fault. Internal faults are called vulnerabilities, whereas external faults are called attacks. The *SecuritySolutions* contains means for preventing unwanted behaviour and development of a system. The last concept is *SecurityMetrics* divided to *StrengthMetrics* and *WeaknessMetrics* – containing eight metrics to measure system's threats and efficiency of their countermeasures [7]. However, this categorization is at very high abstraction level.

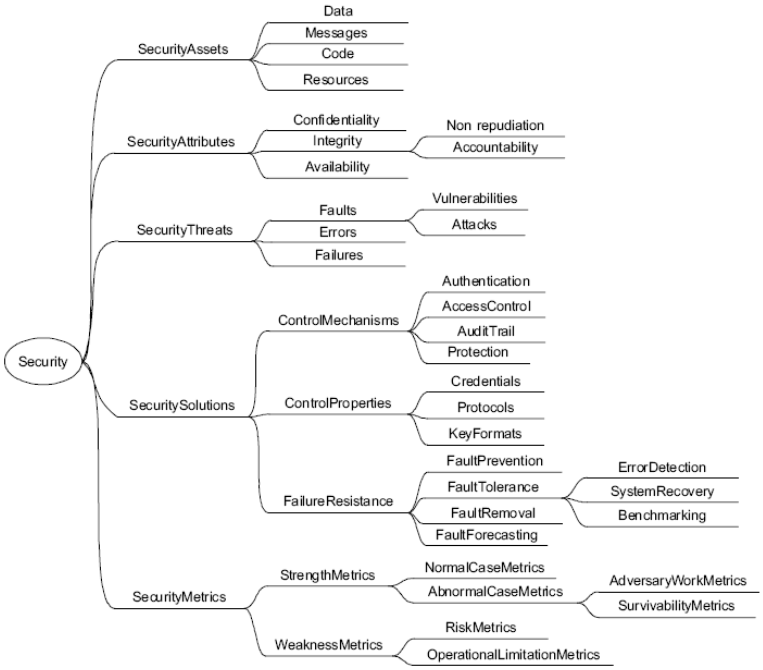


Fig. 7 Security taxonomy by Savolainen et al.

Denker et al. describe security annotations, represented by DAML-S, intended to be used by agents in [8]. Their ontology can be used to describe service requirements and capabilities – e.g. the requirement that a service requestor wants to use the Open-PGP encryption – and this information are used for service discovery and matchmaking. Matchmaking is performed based on requirements and capabilities with four matching level. [8]

Fig 8 presents the main parts of ontology from Denker et al. [8]. These ontologies contain *credentials information* and *security mechanisms*. Therefore, many important security aspects are missing. When compared to the work by Savolainen et al., for instance assets, threats and metrics are not defined. However, purpose of the Denker's ontology is to concentrate mostly to the service discovery and matchmaking, and thus scope is different than ontology in [7]. Lately, Denker et al. updated their ontology to utilising OWL in [9].

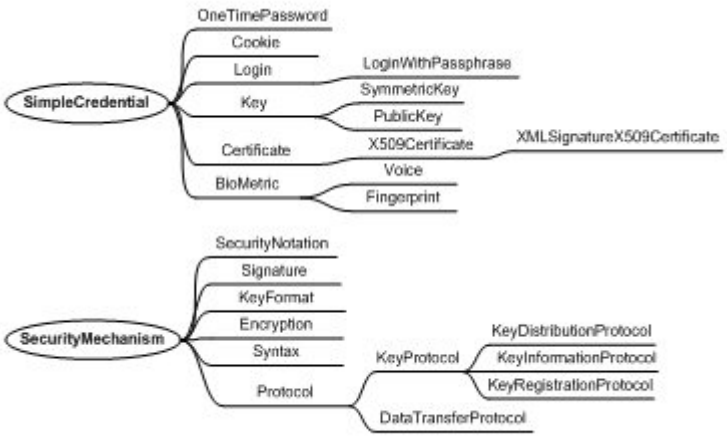


Fig 8 Security credentials and mechanisms by Denker et al.

Kim et al. presented their security ontology called NRL (Naval Research Laboratory) security ontology in [10]. NRL security ontology makes it possible to annotate resources with security related information that can be used during service discovery and matchmaking. NRL security ontology describes following security aspects: *mechanisms, protocols, objectives, algorithms* and *credentials* in various levels of details. Both service providers and requestors can use NRL ontology to describe their capabilities and requirements. Actually, NRL security ontology is a collection of ontologies. Main Security ontology imports the Credentials, Security Algorithms and Security Assurance ontologies as object properties. Thus, these ontologies make it possible to give more specific values for *SecurityConcepts*. In addition, user can define *SecurityObjectives* and connect them to the specific *SecurityConcept* by utilising *supportSecurityObjective* property. [10]

NRL security ontology is well organised concentrating mostly on security solutions area and providing a reasonable way to matchmaking between requirements and capabilities. When comparing NRL ontology to work by Savolainen et al. it can be noticed that the NRL security ontology lacks in security assets, threats and metric areas. However, NRL security ontology contains a substantially better description in security solutions area. In addition, connections between security objectives (requirements) and their solutions are defined more comprehensively.

Tsoumas et al. present in [11] and [12] a framework for information system security management, i.e. linking high-level policy statements to explicit low-level security controls adaptable and applicable in the information system environment. Authors' framework consists of four phases: 1) Building of Security Ontology, 2) Security Requirements Collection, 3) Security Actions Definition, and 4) Security Actions Deployment and Monitoring. Building of Security Ontology means ontology's instantiation based on a conceptual model. This conceptual model defines: *Asset*, *Attack*, *Controls*, *Countermeasure*, *Impact*, *Security policy*, *Stakeholder*, *Risk*, *Threat*, *Threat agent*, *Unwanted incident*, *Vulnerability* and *relationships* among these concepts. When compared ontology from Tsoumas et al. to ontology from Savolainen et al. it can be noticed that both ontologies describe security in the same abstraction level. Thus, neither describes detailed protocols or mechanism for achieving security as opposed to NRL security ontology does. As well as NRL security ontology and ontology from Denker et al., ontology from Tsoumas et al. suffers from lack of security metrics.

Table 2 summarizes the most important aspects of above described security ontologies from run-time security management viewpoint. Ontologies from Denker et al. and Kim et al. are initially intended for run-time environment, i.e. service discovery and matchmaking. On the other hand, only ontology from Savolainen et al. contains security metrics. Thus, combining proposals from Savolainen et al. and Kim et al. should offer an appropriate approach. However, combining ontologies in an appropriate way contains many challenges. Firstly, suitable abstraction level has to be found. Secondly, relationships between metrics and security solutions are required, in order to measure solutions' efficiency during the run-time. In addition one of the major challenges is how to incorporate attacker behaviour to the ontology. Malicious activity is the major difference in security compared to other quality attribute descriptions.

Table 2 Summary of security ontologies

Ontology	Scope	Pros / cons
Savolainen et al.	Design phase of service centric systems.	+ Contains metrics – No connection between attributes and solutions – Only abstract level hierarchy – Intended for design-time use
Denker et al.	Service discovery and matchmaking	+ Run-time applicable – Only credentials and mechanisms.
Kim et al.	Service discovery and matchmaking	+ Run-time applicable + Detailed solutions + Connection between objectives and solutions – Metrics are missing
Tsoumas et al.	Linking policy statements to security controls	– Only abstract level hierarchy – Intended for design-time use

2.2 Security Measurement

It is a widely accepted management principle that an activity cannot be managed well if it cannot be measured. Overall, metrics provide four fundamental benefits – to characterize, to evaluate, to predict and to improve. Security metrics and measurements can be used for decision support, especially in assessment and prediction. Examples of using security metrics for assessment include [14]:

- Comparison of different security controls or solutions,
- Security assurance of a product, an organization, or a process,
- Security testing (functional, red team and penetration testing) of a system,
- Certification and evaluation (e.g. based on Common Criteria [15]) of a product or an organization,
- Intrusion detection in a system, and
- Other reactive security solutions such as antivirus software.

The field of defining security metrics systematically is young and the current practice of information security is still a highly diverse field; holistic and widely accepted approaches are still missing. A major challenge in developing appropriate and feasible security metrics is the immaturity of the state-of-the-art security requirements engineering. Security requirements have not been profoundly addressed within the software engineering community: they are still regarded as being in a side role in most of the software requirements engineering codes of practice [18].

In order to measure, the target of measurement needs to be identified. It is important to clearly know the entity that is the target of measurement because otherwise the actual metrics might not be meaningful. The target of security measurement can be, e.g., an organization, its processes and resources, or a product or its subsystem. The most widely known technical security certification standard is the Common Criteria (CC) ISO/IEC 15408 international standard [15]. During the CC evaluation process, a numerical rating, EAL (Evaluation Assurance Level), is assigned to the target product, with EAL1 being the most basic and EAL7 being the most stringent level. Each EAL corresponds to a collection of assurance requirements, which covers the complete development of a product with a given level of strictness.

ISO/IEC 9126 standard concentrates measuring software's quality, and it contains three metric reports: 1) External metrics [4] applicable during testing, 2) Internal metrics [5] applicable during development and 3) Quality in use metrics [6] applicable in run-time. Furthermore, ISO/IEC divides quality attributes into characteristics and subcharacteristics – security can be found under the functionality characteristic. Therefore, metrics from ISO/IEC are grouped to these characteristics and their subcharacteristics. However, standard does not contain any security related metric in Quality in use metric document [6].

Therefore, it is necessary to find a suitable way to measure security more extensively and holistically. Wang et al. presented a security measurement framework in [2] based on security requirements. The main idea in their work is to divide security requirements to smaller parts, i.e. decomposition, and finally these smallest parts can be measured. Garcia et al. propose two terms: a base measure and a derived measure in [19]. The base measure is a measure of quality attribute that does not depend on other measures, whereas the derived measure is a measure derived from base or derived measures. Therefore, base and derived

measure terms can be combined to the approach of Wang et al. **Fig 9** shows the decomposition made for non-repudiation in [2] – combined with the base measure and derived measure terms from [19]. It can be noticed that the lowest level in **Fig 9** contains reliability – and metrics for it can be found, for example from ISO/IEC's standard. Thus, decomposition gives a possibility to find and develop security metrics required for the run-time security monitoring. When searching these metrics following sources can also give a valuable input: The U.S. NIST (National Institute of Information Standards and Technology) SAMATE (Software Assurance Metrics and Tool Evaluation) project [16] that seeks to help answer various questions on software assurance, tools and metrics. OWASP [17] contains an active discussion and development forum on security metrics. More security metrics are listed in [14] and [18]. Representing these base and derived measures in the ontology also makes it possible to utilise measures at run-time, and in addition, generate new derived measures from the existing measures if needed.

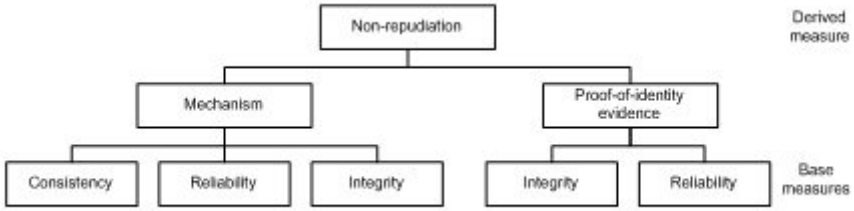


Fig 9 Decomposition for non-repudiation

3 Security Modelling

This section describes how to design security and how the process should take run-time security management issues into account in order to make it possible to achieve run-time security management.

3.1 Design-time Security Management

Our earlier work has concentrated to take quality issues into account at software's design and implementation phases by exploiting architectural models [20] and [21], i.e. conforming to model-driven development. We have used ontologies to describe quality attributes in a uniform way. Ontologies are utilized during the quality requirements modelling and during the architecture modelling to select the best solutions to achieve required qualities. Furthermore, we have evaluated a designed architecture in order to detect whether required qualities are met or not. Finally, quality of the implemented software is measured and compared to requirements. All of these phases belong to the QADA (Quality-driven Architecture Design and quality Analysis) methodology [23]. QADA contains two abstraction levels, i.e. conceptual and concrete levels, which can be mapped to the PIM (Platform-Independent Model) and PSM (Platform-Specific Model) from MDA.

Based on our earlier work, we are able to take step forward and start to concentrate to the run-time quality management – especially from security point of view. The means for defining quality attributes (especially reliability) at modelling-time is represented in [20] – and **Fig. 10** follows this approach from the security viewpoint. Thus at this point, the purpose is to model security requirements and the architecture in a way that makes it possible to achieve run-time security management.

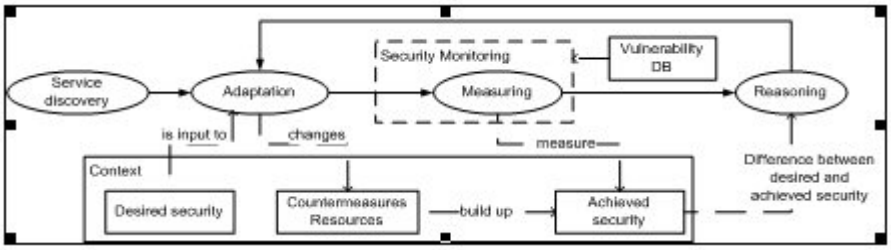


Fig. 10 Main phases of security modelling process.

Firstly, ontologies work as an input for the *Defining requirements* activity. Thus, ontologies described in the previous section have to be combined appropriately in order to achieve one adequate ontology – containing at least security objectives, countermeasures, base/derived measures and relationships between these concepts. After that, the security ontology can facilitate a requirements definition activity comprehensively, as a guideline. For instance, an architect can define following security requirements based on security objectives described in the security ontology and the objectives of the designed software: "user has to be authenticated" and "messages' integrity has to be ensured".

Constructing architecture activity starts after requirements definition. Support for run-time security adaptation requires that alternative security solutions are modelled and implemented into the system. The security ontology helps to find these alternative security solutions. Therefore, the architect selects from the security ontology a password and fingerprints as alternative solutions for the authentication requirement and AES (Advanced Encryption Standard) and DES (Data Encryption Standard) as alternative solutions for the integrity requirement. In addition, the architect selects appropriate measures from the security ontology to monitor achievement of the authentication and integrity requirements. Finally, we have security annotated architecture that works as an input for *Implementation* activity. Therefore, implementation can produce software containing alternative solutions for achieving security on different contexts, and measurements for ensuring that desired security level is kept.

In conclusion, the security ontology is used both defining security requirements and constructing architecture that contains alternative security solutions. Without these steps it is not possible to achieve run-time security management described in the next section.

3.2 Run-Time Security Management

Our current vision of the run-time security management is presented in **Fig. 11** – in other words, how the above designed and developed software monitors and adapts its security.

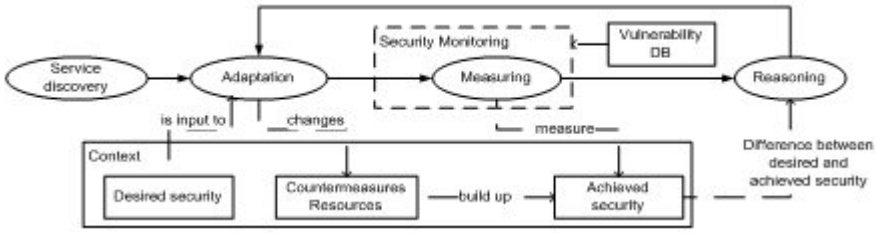


Fig. 11 Run-time security management

Firstly, available services are discovered. Thereafter, *Adaptation* is performed based on available services, their security properties, and the desired security level. In this phase, the security ontology helps to select the most suitable countermeasure (i.e. security solution as mentioned in the previous section) and resources in order to achieve the desired security level. Furthermore, this phase can utilize the existing service matchmaking solutions, but the adaptation also requires more sophisticated algorithms for the decision making. The purpose is to achieve an automatic adaptation that does not require user actions.

However, user preferences can be used to direct a countermeasure selection or setting a divergent desired security level.

Next, the achieved security is *monitored* by means of measuring. A *Measuring* activity measures several properties of the system utilising base measures – introduced in **Fig 9**. From these values the *Monitoring* activity combines an overall security level by means of derived measures. Thus, monitoring activity also requires security ontology. Vulnerability databases can be utilised as an additional input source for monitoring activity. For instance, selected countermeasure might be cracked after ontology construction, and thus its security efficiency is lower than initially set. Hence, utilisation of vulnerability DBs enhances correctness of monitoring.

The achieved security value acts as an input for the *Reasoning* activity, which calls the *Adaptation* if the achieved security level is not satisfying the desired security and alternative countermeasure can be selected. Both the monitoring and the reasoning activities should be automatic – working without user actions. On the other hand, if the desired security level is not achieved and adaptation cannot resolve a situation then user actions will be needed to make a decision how to continue. In **Fig. 11** desired and achieved security levels are part of the context because the security management depends on the context where the adaptation occurs. How context is to be defined is out of the scope of this paper.

As a simple example of run-time security management, software contains AES / DES encryptions and fingerprint / username password pair authentication for achieving its security – as implemented in the previous section. We assume that a user prefers to use fingerprint authentication without encryption as default. In the case when content of the user's information exchange changes, for example, from news reading to more secure online buying, i.e. context changes, the desired security level also changes. The monitoring activity uses base and derived measures from the security ontology for monitoring the achieved security level. Based on the results of the monitoring, the reasoning activity remarks that the desired security level is not reached anymore. Hence, the adaptation is called, and it decides that the security level for online buying cannot be satisfied without encryption and thus AES is selected – for instance.

4 Conclusions and Future Work

Managing security at the run-time requires that: 1) security is understood holistically yet offering enough details to be useful, 2) the achieved security level of the software can be monitored, and 3) software is able to adapt itself based on the context and monitoring results. In this paper, we concentrated to the first issue and also presented the overview vision how the run-time security management can be achieved.

Several security ontologies exist but mostly emphasize service discovery and matchmaking, or alternatively describe different security solutions. Although both are important aspects, these are not enough for covering the whole area of security from the run-time point of view. Thus, there is a need for more extensive security ontology. In addition, security ontology has to contain base and derived measures which make it possible to measure and monitor security levels.

Our next step is to produce the universal security ontology – a combination from the existing ones – containing security objectives and supporting solution mechanisms. Next, we will develop a set of base and derived measures for measuring efficiency of security solutions and include these metrics to the combined ontology. After that we are able to move forward to research monitoring mechanisms and adaptation algorithms required to support the whole process. When the first monitoring mechanisms are available we can build up an initial laboratory case to test validity of the approach and reveal it costs related the performance and other quality attributes.

Acknowledgments. This work is made under SOFIA (Smart Objects For Intelligent Applications) project – funded by Tekes (the Finnish Funding Agency for Technology and Innovation) and the European Commission.

References

1. Avižienis, A., Laprie, J-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing. Volume 1, Issue 1, pp. 11-33 (2004)

2. Wang, C., Wulf, W. A.: Towards a framework for security measurement. 20th National Information Systems Security Conference, Baltimore, pp. 522-533 (1997)
3. ISO/IEC: 9126-1 Software engineering – Product quality – Part 1: Quality model (2001)
4. ISO/IEC: 9126-2 Software engineering – Product quality – Part 2: External metrics (2003)
5. ISO/IEC: 9126-3 Software engineering – Product quality – Part 3: Internal metrics (2003)
6. ISO/IEC: 9126-4 Software engineering – Product quality – Part 4: Quality in use metrics (2004)
7. Savolainen, P., Niemelä, E., Savola, R.: A Taxonomy of Information Security for Service-Centric Systems. 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 5--12 (2007)
8. Denker, G., Kagal, L., Finin, T., Paulucci, M., Sycara, K.: Security for DAML web services: Annotating and matchmaking. In Proc. of the 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, pp. 335-350 (2003)
9. Denker, G., Kagal, L., Finin, T.: Security in the Semantic Web Using OWL. Information Security Technical Report, Volume 10, Issue 1, pp. 51-58 (2005)
10. Kim, A., Luo, J., Kang, M.: Security Ontology for Annotating Resources. OTM Confederated International Conferences, CoopIS, DOA, and ODBASE. Springer, Heidelberg. pp. 1483--1499 (2005)
11. Tsoumas, B., Dritsas, S., Gritzalis, D.: An Ontology-Based Approach to Information Systems Security Management. In Computer Network Security, pp. 151-164 (2005)
12. Tsoumas, B., Gritzalis, D.: Towards an Ontology-Based Security Management. 20th International Conference on Advanced Information Networking and Applications AINA), pp. 985-992 (2006)
14. Savola, R.: Towards a Taxonomy for Information Security Metrics. In Proc. of ACM Workshop of Quality of Protection (QoP'07), Alexandria, Virginia, USA, pp. 28-30 (2007)
15. ISO/IEC: 15408-1 Common Criteria for Information Technology Security Evaluation – Part 1: Introduction and General Model. (2005)
16. Black, P. E.: SAMATE's Contribution to Information Assurance. IANewsletter, Volume 9, Issue 2 (2006)
17. OWASP: Open Web Application Security Project. http://www.owasp.org/
18. Savola, R., Abie, H.: Identification of Basic Measurable Security Components for a Distributed Messaging System. 3rd Int. Conf. on Emerging Security Information, Systems and Technologies (SECURWARE), Jun 18-23, 2009, Athens, Greece (2009) in press
19. Garcia, F., Bertoa, M. F., Calero, C., Vallecillo, A., Ruíz, F., Genero M.: Towards a consistent terminology for software measurement. In Information and Software Technology, Volume 48, Issue 8, pp. 631-644 (2006)
20. Niemelä, E., Evesti, A., Savolainen, P.: Modeling Quality Attribute Variability: 3rd international conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 169-176 (2008)
21. Evesti, A., Niemelä, E., Henttonen, K., Palviainen, M.: A Tool Chain for Quality-driven Software Architecting. 12th International Software Product Line Conference (SPLC), p. 360 (2008)
22. Anderson, R.: Security Engineering – A Guide to Building Dependable Distributed Systems. John Wiley & Sons, New York (2001)
23. QADA (Quality-driven Architecture Design and quality Analysis). www.vtt.fi/qada/

Automatic Generation of Security-Aware GUI Models

Michael Schläpfer[1], Marina Egea[1], David Basin[1], and Manuel Clavel[2,3]

[1] ETH Zürich, Switzerland
{basin,marinae}@inf.ethz.ch,michschl@student.ethz.ch
[2] IMDEA Software Institute, Madrid, Spain
manuel.clavel@imdea.org
[3] Universidad Complutense de Madrid, Spain
clavel@sip.ucm.es

Abstract. In typical software applications, users access application data using GUI widgets. There is an important, but little explored, link between visualization and security: when the application data is protected by an access-control policy, the application GUI should be *aware of* and *respect* this policy. For example, a widget should not give users options to execute actions on the application data that they are not authorized to execute. However, GUI designers are not (and usually should not be) aware of the application data security policy. To solve this problem, we define in this paper a many-models-to-model transformation that, given a security-aware data model and a GUI model, makes the GUI model also security-aware.

1 Introduction

In typical software applications, users access application data using GUI widgets: data is created, deleted, read, and updated using text boxes, check boxes, combo boxes, buttons, and the like. There is an important, but little explored, link between visualization and security: When the application data is protected by an access-control policy, the application GUI should be *aware of* and *respect* this policy. Otherwise, users will often experience frustration. For example, after filling out a long electronic form, the user may be informed that the form cannot be submitted because she lacks permissions to execute the actions that are required on the application data. However, the GUI designers are not (and usually should not be) aware of the application data security policy. Their job is simply to design the GUI's layout and to specify its behaviour, i.e., which events will trigger which actions on which application data and/or application widgets.

To solve this problem, we define in this paper a many-models-to-model transformation that, given a security-aware data model and a GUI model, makes the GUI model also security-aware. This model transformation is the key component of our proposal for designing security-aware application GUI models. Figure 1 illustrates this proposal. The process of designing a security-aware GUI has the following parts. First, software engineers specify the application data model M.

Then, security engineers specify, in the security model $S(M)$, the application-data access-control policy, and GUI designers specify the application GUI model $G(M)$. Finally, the application security-aware GUI model $S(G(M))$ is automatically generated from the security model $S(M)$ and the GUI model $G(M)$.

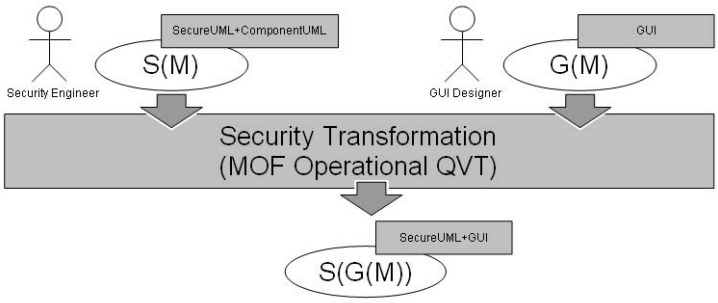


Fig. 1. Generating security-aware application GUIs.

In Section 2 we introduce the source and target models of the transformation by describing their respective metamodels (namely, SecureUML+ComponentUML, GUI, and SecureUML+GUI). Then, in Section 3, we describe the transformation as a QVT operational transformation and, in Section 4, we discuss its correctness. Finally, in Section 5, we give an overview of the planned extensions of the ideas presented in this paper. We illustrate our ideas on a running example, namely, the design of a simple GUI for a phone-book application. As part of our work, we have implemented the transformation using the Operational QVT transformation engine that is provided within the M2M Project, a subproject of the Eclipse Modeling Framework. Our tool is available at [15] along with documentation and examples.

Our model-transformation based approach for designing security-aware GUI models has three principle advantages over traditional software development approaches.

1. Security engineers and GUI designers can independently model what they know best (or know at all).
2. Security engineers and GUI designers can independently change their models and these changes are automatically propagated to the final security-aware GUI models.
3. GUI designers, even if they do not know the underlying security policy, can still check its impact on their designs. They can use the final security-aware GUI models to check that they are designing the right GUI to give the (authorized) users access to the (intended) application data.

Our proposal for automatically generating security-aware GUI models is the corner stone of a more ambitious project for making model-driven security an

effective and useful approach for generating multiple system layers as part of a security-intensive industrial software development. A crucial property of these systems that we directly pursue with our model-transformation based approach, is *conformance*. For the case addressed in this paper, conformance means that executing events on the GUI layer never leads to program exceptions from the access-control security policy implemented at the persistent layer.

2 The Transformation Model Types

Model transformation is the process of converting some models $M_1, \ldots, M_n$ (the transformation *source* models) into other models $M'_1, \ldots, M'_m$ (the transformation *target* models). In this section, we define the *types* of the source and target models of our model transformation by introducing their respective metamodels: namely, the SecureUML+ComponentUML metamodel and the GUI metamodel (which define our source models' types), and the SecureUML+GUI metamodel (which defines our target model's type). Since SecureUML+ComponentUML and SecureUML+GUI are both dialects of SecureUML, we begin this section by briefly introducing SecureUML.

The SecureUML language. SecureUML is a language based on RBAC [6] for modeling access-control policies on protected resources [2]. The policies that can be specified in SecureUML are of two kinds: those that depend on static information, namely the assignments of users and permissions to roles; and those that depend on dynamic information, namely the satisfaction of authorization constraints in the current system state. However, SecureUML leaves open what the protected resources are and which actions they offer to clients. These are specified in a so-called *dialect* and depend on the primitives for constructing models in each dialect's system-design modeling language. Figure 2 shows the SecureUML metamodel: each SecureUML dialect will basically declare its own protected resources as subclasses of *Resources* and the actions that they offer to clients as subclasses of *Atomic* or *Composite Actions*.

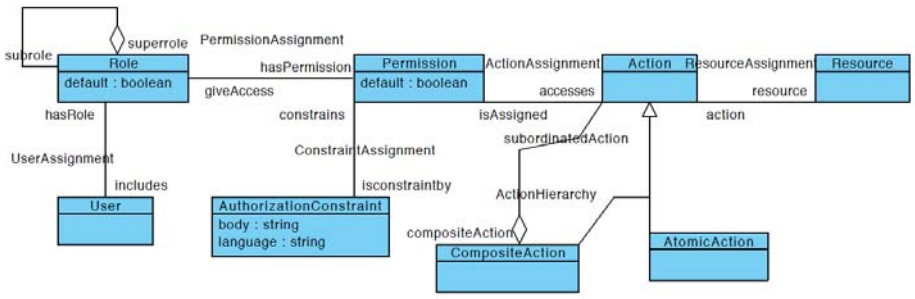


Fig. 2. The SecureUML metamodel.

2.1 The source model types

The source models of our many-models-to-model transformation are SecureUML+ComponentUML models and GUI models. In a nutshell, the former specify the policies for accessing the application data, while the latter specify which of the actions on the application data are triggered by which application GUI events. Importantly, our source models have the same underlying application-data model.

The SecureUML+ComponentUML language. This language combines SecureUML with ComponentUML, which is a simple language for modeling component-based systems. ComponentUML provides a subset of UML class models: *Entities* can be related by *Associations* and may have *Attributes* and *Methods*. The SecureUML+ComponentUML metamodel, partially shown in Figure 3, provides the connection between SecureUML and ComponentUML. It specifies the following.

- The protected resources, namely, *Entities*, as well as their *Attributes*, *Methods*, and *AssociationEnds* (but not *Associations* as such).
- The actions on these protected resources and their hierarchies. These are shown in the following table.

Resource	Actions
Entity	create, read, update, delete, full access
Attribute	read, update, full access
Method	execute
Association end	read, update, full access

In this table, composite actions are underlined. They are used to group primitive actions into a hierarchy of higher-level ones: e.g., full access on an attribute includes both read and update access on this attribute, and full access on an entity includes both full access on the entity attributes, entity methods, and methods for entity creation and deletion.

In [2] a UML profile is defined for drawing SecureUML+ComponentUML models, which we summarize here. A role is represented by a UML class with the stereotype $\langle\langle Role \rangle\rangle$ and an inheritance relationship between two roles is defined using a UML generalization relationship. The role referenced by the arrowhead of the generalization relationship is considered to be the superrole of the role referenced by the tail. A permission, along with its relations to roles and actions, is defined in a single UML model element, namely an association class with the stereotype $\langle\langle Permission \rangle\rangle$. The association class connects a role with a UML class representing a protected resource, which is designated as the root resource of the permission. The actions that such a permission refers to may be actions on the root resource or on subresources of the root resource. Each attribute of the association class represents the assignment of an action to the permission, where the action is identified by the name and the type of the attribute. ComponentUML entities are represented by UML classes with

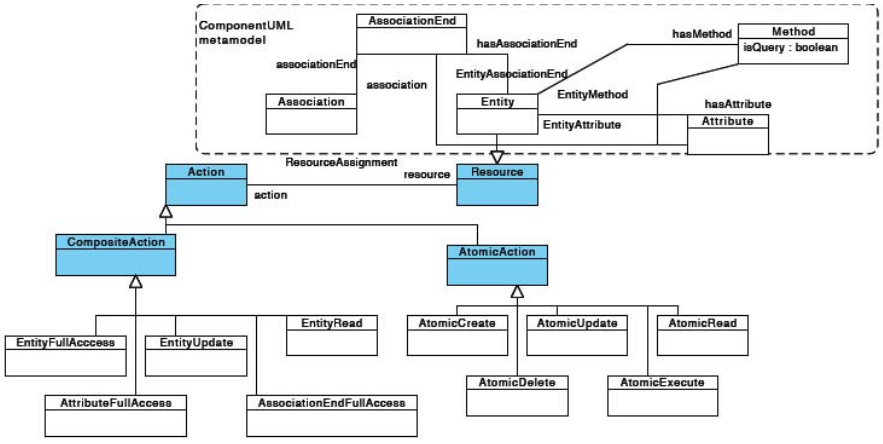


Fig. 3. The SecureUML+ComponentUML metamodel (partial).

the stereotype $\langle\langle Entity \rangle\rangle$. Every method, attribute, or association end owned by such a class is automatically considered to be a method, attribute, or association end of the entity.

Example 1. Consider a basic phone-book application, called PhoneBook. Each entry in the underlying phone directory consists of a name and a phone number. The access to this data is controlled by the following policy:

- Users are only allowed to read people's name and phone numbers.
- Supervisors are allowed to read people's name and phone numbers, as well as to change phone numbers.
- Administrators are allowed to create and delete entries in the phone directory, as well as to read and write them.

Figure 4 shows the SecureUML+ComponentUML model that specifies the above policy.

The GUI language. We now introduce a simple language for modeling GUIs, which is however rich enough for our present purposes. Its metamodel is shown in Figure 5. Application GUIs consist of *Widgets* that are displayed inside *Containers*, which are themselves *Widgets*. Each widget has a (possibly empty) set of *ActionEvents* associated to it, which specifies how the widget reacts to events. For the purpose of this paper, we can assume that each event can only trigger one action on the application data.[4] We also assume that, in each instance of *ActionEvent*, the value of the attribute *modelAction* is a SecureUML+ComponentUML atomic action and that this action has as its root resource an entity declared in the ComponentUML model that specifies the underlying application-data model.

[4] In general, one event can trigger many actions, some of them acting on the application data (create, delete, read, update, or execute) and some of them acting on the GUI widgets (close, open, hide, and so on).

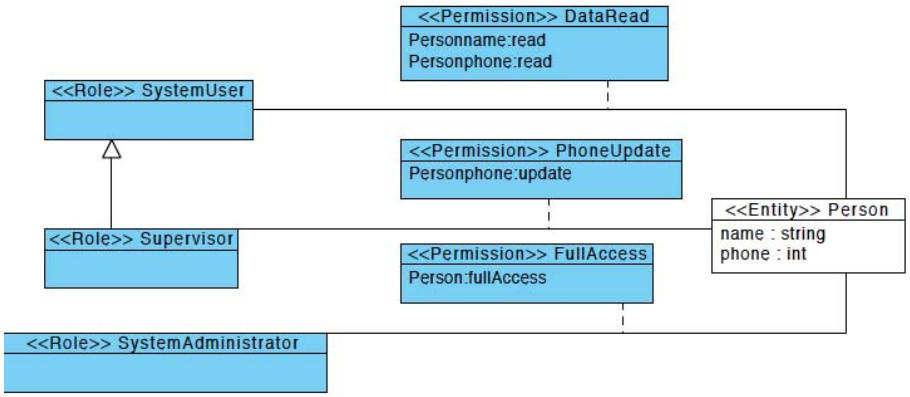


Fig. 4. A simple security policy for a PhoneBook application.

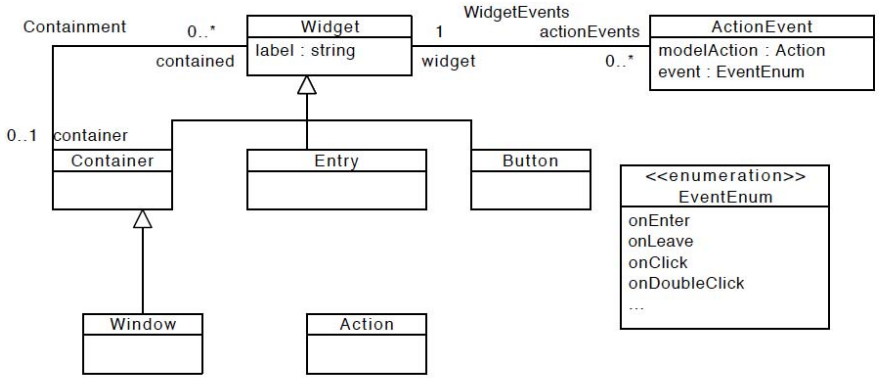


Fig. 5. A simple metamodel for GUIs.

Example 2. Suppose that our PhoneBook GUI designer decides that the application should provide a basic interface for editing entries in the directory. She therefore designs a GUI consisting of a window *PhoneBook Editor*, with two entry boxes: *Name* and *Phone Number*. She also decides that, at the time of creation, each instance of *PhoneBook Editor* is associated to an instance P of the entity *Person*. Moreover, the GUI should offer the following functionality.

- *On entering* the entry box *Name*, the box should display the name of the object P, i.e., she associates the event *onEnter* with a *read* action on the attribute *name* of an instance of the entity *Person*. Similar actions should occur when entering the entry box *Phone Number*.
- *On leaving* the entry box *Name*, the text currently displayed in this box should be used to update the attribute *Name* of the person P, i.e., she associates the event *onLeave* with an *update* action on the attribute *name* of

an instance of the entity *Person*. Similar actions should occur when leaving the entry box *Phone Number*.

Figure 6 shows the instance of the GUI metamodel that corresponds to the GUI model specifying the above GUI.

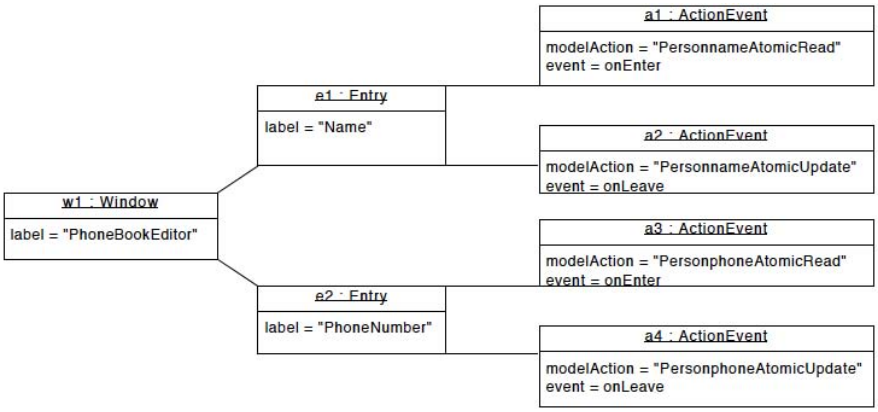


Fig. 6. A simple GUI for editing PhoneBook entries.

2.2 The target model type

SecureUML+GUI models are the target models of our many-models-to-model transformation. SecureUML+GUI combines SecureUML with the GUI modeling language introduced in the previous section. In a nutshell, SecureUML+GUI models specify who can execute which events on which widgets. The SecureUML+GUI metamodel, (partially) shown in Figure 7, provides the connection between SecureUML and GUI. It specifies the following.

- The protected resources, namely, *ActionEvents*.
- The actions on these protected resources, namely, *AtomicExecute*.

In the next section we will show the SecureUML+GUI model that results from applying our transformation to the SecureUML+ComponentUML model and the GUI model discussed, respectively, in Examples 1 and 2.

3 The Transformation Description

We are ready to describe a many-models-to-model transformation that automatically generates SecureUML+GUI models from SecureUML+ComponentUML models and GUI models. We assume here, as explained above, that the

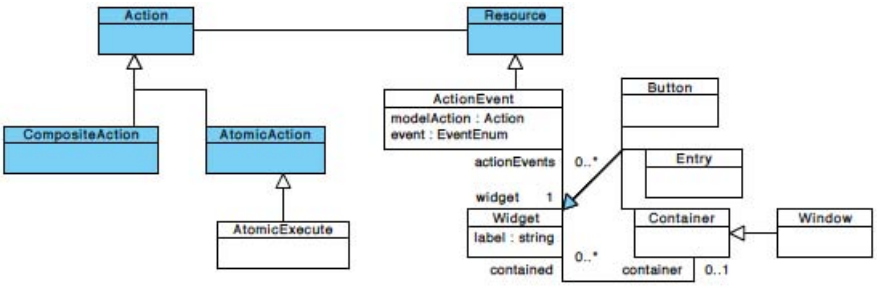


Fig. 7. The SecureUML+GUI metamodel (partial).

source models have the same ComponentUML application-data model. As we will discuss in the next section, our transformation satisfies the expected property, namely, that the (target) SecureUML+GUI model *preserves* the security policy specified in the (source) SecureUML+ComponentUML model.

We now introduce our many-models-to-model transformation as a QVT operational transformation [10, Section 8.4.6] using Operational QVT syntax. In Figure 8 we show the heading of this operational transformation. The metamodels `GUI`, `SECUMLANDCOMPUML`, and `SECUMLADGUI` are the GUI metamodel, the SecureUML+ComponentUML metamodel, and the SecureUML+GUI metamodel that were introduced in Section 2; their definitions are available at [15]. The operational transformation is defined by mapping functions, which are executed sequentially. Due to space limitations, we can only describe these functions here; their full definitions are also available at [15]. The final target models are obtained in two steps.

Step 1: The model elements of the target model are created as follows.

– The *Roles* in the (source) SecureUML+ComponentUML model are copied, along with their hierarchies, in the (target) SecureUML+GUI model, using the following mapping functions.

```
secPolicy.objects()[SECUMLANDCOMPUML::Role]->map Role_to_Role();
secPolicy.objects()[SECUMLANDCOMPUML::Role]->map
              preserve_Role_hierarchy();
```

– The *Widgets* in the (source) GUI model are copied, along with their containment relationships and their associated *ActionEvents*, in the (target) SecureUML+GUI model, using the following mapping functions.

```
guiModel.objects()[GUI::ActionEvent]->map
             ActionEvent_to_ActionEvent();
guiModel.objects()[GUI::Window]->map Widget_to_Widget();
guiModel.objects()[GUI::Entry]->map Widget_to_Widget();
guiModel.objects()[GUI::Button]->map Widget_to_Widget();
```

```
guiModel.objects()[GUI::Widget]->map
              preserve_containment_hierarchy();
```

– For each *ActionEvent* in the (source) GUI model, an *AtomicExecute* action is created and linked to the *ActionEvent* as to its root resource in the (target) SecureUML+GUI model using the following mapping function.

```
guiSecPolicy.objects()[SECUMLANDGUI::ActionEvent]->map
               addAtomicExecuteAction();
```

Step 2: The permission assignment in the target model are created as follows.

– For each *ActionEvent*'s *action* in the (source) GUI model, and for each *Role* that is allowed to perform this *action* in the (source) SecureUML+ComponentUML model, a *Permission* is created in the (target) SecureUML+GUI model that grants access to *Role* to execute the *ActionEvent*'s *event*. This is accomplished using the following mapping function.

```
guiSecPolicy.objects()[SECUMLANDGUI::ActionEvent]->liftPermissions();
```

```
modeltype GUI uses "http://gui/1.0";
modeltype SECUMLANDCOMPUML uses "http://secumlandcompuml/1.0";
modeltype SECUMLANDGUI uses "http://secumlandgui/1.0";

transformation SecurityTransformation(in guiModel : GUI,
                in secPolicy : SECUMLANDCOMPUML,
                out guiSecPolicy : SECUMLANDGUI);

main() {
  /* Step 1: Creating the model elements of the target model */
  secPolicy.objects()[SECUMLANDCOMPUML::Role]->map Role_to_Role();
  secPolicy.objects()[SECUMLANDCOMPUML::Role]->map
                preserve_Role_hierarchy();
  guiModel.objects()[GUI::ActionEvent]->map
                ActionEvent_to_ActionEvent();
  guiModel.objects()[GUI::Window]->map Widget_to_Widget();
  guiModel.objects()[GUI::Entry]->map Widget_to_Widget();
  guiModel.objects()[GUI::Button]->map Widget_to_Widget();
  guiModel.objects()[GUI::Widget]->map
                preserve_containment_hierarchy();
  guiSecPolicy.objects()[SECUMLANDGUI::ActionEvent]->map
                addAtomicExecuteAction();

  /* Step 2: Creating permission assignments in the target model */
  guiSecPolicy.objects()[SECUMLANDGUI::ActionEvent]->liftPermissions();
```

Fig. 8. The many-models-to-model transformation's heading in QVTO syntax.

Example 3. We show in Figure 9 the (relevant aspects of the) SecureUML+GUI model that results from applying our many-models-to-model transformation to the SecureUML+ComponentUML model and the GUI model discussed, respectively, in Examples 1 and 2. Here, to draw our resulting SecureUML+GUI model, we use a UML profile similar to the one available for SecureUML+ComponentUML models, except that the stereotype $\langle\langle Entity \rangle\rangle$ is now reserved for *Widgets*, which contains as methods their associated *ActionEvents*' *events*.

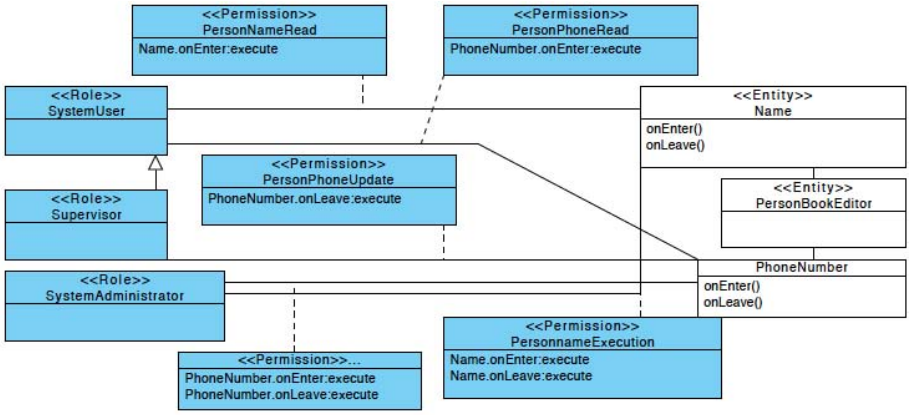


Fig. 9. A security-aware simple GUI for editing PhoneBook entries.

Interestingly, a simple analysis of the resulting SecureUML+GUI model in Example 3 reveals that only *Administrators* should be allowed to open a *PhoneBook Editor* window, since only they can execute *all* the events associated to the widgets contained in a *PhoneBook Editor* window (in [1] it is explained how SecureUML models can be automatically analysed using OCL queries). Of course, this information is crucial for the GUI designer in order to validate their GUIs, i.e., to check that she is designing the right graphical interface to give the (authorized) users access to the (intended) application data. She may realize, for example, that, in order to give *Supervisors* access to editing peoples' phone number, another GUI is needed with no action associated this time to the event of leaving the *Name* entry box,

4 The correctness of the transformation

In this section, we discuss the correctness of our transformation. Our claim is that the (target) SecureUML+GUI model $S(G(M))$ *preserves* the security policy specified in the (source) SecureUML+ComponentUML model $S(M)$. More specifically, we claim that a role is allowed to execute an event in the $S(G(M))$

model only if it is allowed to execute the action that is associated to this event in the $S(M)$ model.

In [1] we proposed a metamodel-based approach for automatically analyzing security-design models in a semantically precise and meaningful way. More concretely, we showed that security properties of security-design models can be expressed as formulas in OCL [11], the Object Constraint Language of UML. We can formalize queries about the relationships between users, roles, permissions, and actions, and we can answer such queries by evaluating them on the instances of the SecureUML metamodel that represent the security-design model under consideration. We also defined in [1] a number of OCL operators that formalize different aspects of the access-control information contained in the security-design models. In particular, we defined in [1] an operator allAtomics() that, given a role, returns all the atomic actions that a user in this role can perform:

---

context Role::allAtomics():Set(AtomicAction) **body**:
self.allPermissions().allActions()−>asSet()

The operator allPermissions returns the collection of permissions (directly or indirectly) assigned to a role. The operator allActions returns the collection of atomic actions whose access is (directly or indirectly) granted by a permission. We can now use the operator allAtomics() to formally state the correctness property of transformation as follows.

Remark 1. Given a SecureUML+ComponentUML model $S(M)$ and a GUI model $G(M)$, the SecureUML+GUI model $S(G(M))$ resulting from our transformation satisfies the following property: Let *rl* be a *Role* in $S(M)$ and let *acev* be an *ActionEvent* in $G(M)$. Let *ac* be the *Action*-value of *acev*'s attribute *modelAction* in $G(M)$. Suppose now that there exists an *AtomicExecute* action *acex* in $S(G(M))$ that is linked to the *ActionEvent acev* as its root resource. Then, *rl*.allAtomicActions()−>includes(*acex*) evaluates to true in $S(G(M))$ only if *rl*.allAtomicActions()−>includes(*ac*) evaluates to true in $S(M)$.

The above remark follows from the fact that: i) by Step 1 of our transformation, the role hierarchy in $S(G(M))$ is *exactly* the one specified in $S(M)$, and ii) by Step 2 of our transformation, a role is granted permission to execute an action-event in $S(G(M))$ *only if* this role is granted permission to execute in $S(M)$ the action associated to this action-event (as the value of its attribute *modelAction*).

5 The Transformation Extensions

We are currently extending the work presented here in various, related directions. A first direction is to consider application-data security policies that also depend on dynamic information, namely the satisfaction of authorization constraints in the current system state. In this context, our transformation function

must include functions that correctly map authorization constraints in the security models to authorization constraints in the security-aware GUI models. A second direction is to apply our approach to more realistic GUI designs. Here, we will work with GUI models that associate multiple actions, both on the application data and on the GUI widgets, to single events. In this context, Step 2 in our transformation function is less direct. In particular, there could be events whose execution can not be granted to anybody. This would happen if no one is allowed to execute *all* the actions associated to this event. A third direction is to generate GUI models that are not only security-aware, but also *smart*. Smartness is relevant since events can also trigger actions on GUI widgets. For example, in a smart security-aware GUI, widgets should not give users the option to open other widgets when these widgets in turn give them options to execute actions on the application data that they are not authorized to execute. Making security-aware GUIs also smart requires "lifting" permissions in two directions: i) from the widgets whose events triggered actions on application data to the widgets whose events triggered actions on those widgets and ii) from the widgets that are contained in other widgets to the widgets that contain those widgets.

Overall, we aim to provide GUI designers with better models and tools for building and analyzing GUIs for security-critical applications, including tools for automatically checking that, using a given GUI, (authorized) users can indeed access the (intended) application data, or tools for automatically building GUIs intended for specific users, in which all (and only) the (authorized) actions on the application data are indeed accessible by the (intended) users.

6 Related Work

Creating user interfaces is a common task in application development. It can also be very time consuming and therefore expensive. Many proposals have been made, and tools have been built, that aim to reduce the efforts required to build effective and user-friendly graphical interfaces. Surprisingly, despite all these initiatives, until now there has been no research into the systematic design of GUIs whose functionality should adhere to the security policy designed for the underlying application-data model.

In the modeling community, other researchers have investigated how to extend existing modeling environments for GUI modeling. For instance, [5, 3, 4] propose various UML extensions for this purpose. There are also approaches [8] suggesting the use of off-the-shelf web widget libraries to develop web-based user interfaces for semantic web applications, where developers can use RDF constructs to map the data contained in the underlying data model to the model implemented by the widget. More directly related to MDA, [13] reviews the tools that currently support general modeling, model transformations, model weaving, and model constraints in relation to the special needs of the human-computer interaction (HCI) community. Another survey is given in [14], which focuses on transformation tools for model-based user interface development. In relation to security, [16, 17] uses QVT to handle security requirements in an

MDA setting; in particular, to obtain the secure logical scheme from conceptual models. Also, [7] uses the Sectet-framework to integrate security requirements with models at the abstract level and proposes a QVT-based chain of tools that transform these models into artefact's configuring security components of a Web services-based architecture. To the best of our knowledge, none of these approaches is appropriate for modeling application-data access-control security policies at the GUI level. Also, we are not aware of other approaches based on model-transformations for automatically generating security-aware GUI models from security-design models, that is, from models that integrate system designs with their access-control policy.

In the programming community, independent of model-driven initiatives, numerous projects have addressed implementing graphical user interfaces for application data. For example, [9] proposes enriching the application code source with annotations that control the generation of the graphical user interfaces. Other researchers have designed and implemented specialized tools that support the automatic generation of graphical user interfaces meeting their own specific requirements. These tools simplify configuring personal services, enabling the combination of different kinds of events [12]. Also, there are many GUI builders, either integrated in IDEs or available as plug-ins, that simplify the task of creating application GUIs in different programming languages.

7 Conclusions

In this paper we have presented an approach based on model-transformation for automatically generating security-aware GUI models. Given a security-aware data model and a GUI model, our transformation makes the GUI model also security-aware. We have introduced the source and target models of this transformation (by describing their respective metamodels) and we have described the main mapping functions that define the transformation as a QVT operational transformation. We have also discussed the correctness of our transformation. As part of our work, we have implemented this approach using the Operational QVT transformation engine that is provided within the M2M Project, a subproject of the Eclipse Modeling Framework.

Our model-transformation based approach for designing security-aware GUI models has three main advantages over traditional software development approaches. First, security engineers and GUI designers can independently model what they know best. Second, security engineers and GUI designers can independently change their models, and these changes are automatically propagated to the security-aware GUI models. Third, GUI designers can use the security-aware GUI models to check that they are designing the right GUI to give the authorized users access to the intended application data. The transformation presented here is the corner stone of a more ambitious project for making model-driven security an effective and useful approach for generating multiple system layers as part of a security-intensive industrial software development.

References

1. D. Basin, M. Clavel, J. Doser, and M. Egea. Automated analysis of security-design models. *Information and Software Technology, Special issue on Model Based Development for Secure Information Systems*, 2008.
2. D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, 2006.
3. K. Blankenhorn and W. Walter. Extending UML to GUI modeling. `http://www.bitfolge.de/pubs/MC2004_Poster_Blankenhorn.pdf`, 2004.
4. P. Pinheiro da Silva and N. W. Paton. UMLi: The unified modeling language for interactive applications. In *UML 2000 - The Unified Modeling Language. Advancing the standard. Third International Conference*, pages 117–132. Springer, 2000. `http://trust.utep.edu/umli/`.
5. TATA Research Development and Design Center. Heavyweight extension of UML for GUI modeling : A template based approach. `http://www.omg.org/news/meetings/workshops/presentations/uml2001_presentations/10-2_Venkatesh_typesasStereotypes.pdf`, 2001.
6. D. F. Ferraiolo, R. S. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.
7. M. Hafner, M. Alam, and R. Breu. Towards a MOF/QVT-Based domain architecture for Model Driven Security. In *Model Driven Engineering Languages and Systems*, volume 4199 of *LNCS*, pages 275–290. Springer Berlin / Heidelberg, 2006.
8. M. Hildebrand and J. van Ossenbruggen. Configuring semantic web interfaces by data mapping. In Siegfried Handschuh, Tom Heath, and Vinhtuan Thai, editors, *Visual Interfaces to the Social and the Semantic Web (VISSW 2009)*, volume 443, February 2009.
9. J. Jelinek and P. Slavik. GUI generation from annotated source code. In *TAMODIA '04: Proceedings of the 3rd Annual Conference on Task models and Diagrams*, pages 129–136, New York, NY, USA, 2004. ACM.
10. Object Management Group. MOF-Queries, Views and Transformations (QVT)-Final adopted specification. Technical report, OMG, 2005. `www.omg.org/docs/ptc/05-11-01.pdf`.
11. Object Management Group. Object Constraint Language specification. Technical report, OMG, May 2006. OMG document available at `http://www.omg.org`.
12. M. Ogura, H. Mineno, N. Ishikaw, T. Osano, and T. Mizuno. Automatic gui generation for meta-data based pucc sensor gateway. In *Knowledge-Based Intelligent Information and Engineering Systems*, volume 5179 of *LNCS*, pages 159–166. Springer Berlin–Heidelberg, 2008.
13. J.L. Pérez-Medina, S.Dupuy-Chessa, and A.Front. A survey of model driven engineering tools for user interface design. In *Task Models and Diagrams for User Interface Design*, volume 4849 of *LNCS*, pages 84–97. Springer Berlin / Heidelberg, 2007.
14. R.Schaefer. A survey on transformation tools for model based user interface development. In *Human-Computer Interaction. Interaction Design and Usability.*, volume 4550 of *LNCS*, pages 1178–1187. Springer Berlin / Heidelberg, 2007.
15. M. Schläpfer. A tool for generating security-aware GUI models. `http://n.ethz.ch/student/michschl`, 2009.

16. E. Soler, J. Trujillo, E. Fernandez-Medina, and M. Piattini. Application of QVT for the development of secure data warehouses: A case study. In *The Second International Conference on Availability, Reliability and Security, 2007. ARES 2007.*, 2007.
17. E. Soler, J. Trujillo, E. Fernandez-Medina, and M. Piattini. A set of QVT relations to transform PIM to PSM in the design of secure data warehouses. In *ARES '07: Proceedings of the Second International Conference on Availability, Reliability and Security*, pages 644–654, Washington, DC, USA, 2007. IEEE Computer Society.